AD-A238 322

AFOSR-TR- 91 0603

FINAL REPORT

NON–ALGORITHMIC ISSUES IN AUTOMATED COMPUTATIONAL MECHANICS

W. W. Tworzydlo, J. T. Oden, J. M. Bass, J. Combs, S. Sheikh

F49620
TR-89-C-0015

USAF, AFSC
AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
BOLLING AFB, DC

TR-91-09

April 30, 1991

THE COMPUTATIONAL MECHANICS CO., INC.
LAMAR CREST TOWERS
7701 NORTH LAMAR, SUITE 200
AUSTIN, TEXAS 78752
(512) 467-0618

COMCO

(w)

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited N/A (Unclassified)
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) TR-91-09	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Computational Mechanics Co.	6b. OFFICE SYMBOL (If applicable)
6c. ADDRESS (City, State, and ZIP Code) 7701 North Lamar, Suite 200 Austin, TX 78752	7a. NAME OF MONITORING ORGANIZATION AFOSR 7b. ADDRESS (City, State, and ZIP Code) 1200 [illegible] Street [illegible], TX 75202-4399 AFOSR/NA Bolling AFB DC 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFOSR	8b. OFFICE SYMBOL (If applicable) NM
8c. ADDRESS (City, State, and ZIP Code) Bolling AFB Washington DC 20322-6448	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F49620-89-C-0015 10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 61102F; PROJECT NO. 2302; TASK NO. B1; WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Non-Algorithmic Issues in Automated Computational Mechanics (w)

12. PERSONAL AUTHOR(S)
W. W. Tworzydlo, J. T. Oden, J. M. Bass, J. Combs, S. Sheikh

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 3/1/89 TO 2/28/91	April 30, 1991	173

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	finite element method, automated design, knowledge engineering, expert systems, computational mechanics

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The general goal of the project was to study the feasibility of the development of an automatic environment for engineering design of aerospace structures, in particular for their analysis by the finite element method. Of particular interest were non-algorithmic issues related to automated decision making based on heuristics and expertise. In the first phase of work, the types of knowledge and decisions involved in the design process were studied, and computer technologies best suited for their automation were evaluated. These technologies include algorithmic procedures, knowledge-based expert systems, neural networks, knowledge acquisition systems, etc. The main thrust of research in the project was focused on the development and extension of concepts related to automated computational mechanics, such as adaptive computational techniques, automated model and strategy selection, automated performance monitoring and quality assurance for the finite element analysis. These methods were implemented in an automated computational environment, based on full coupling of h-p adaptive finite element program with expert systems technology. Several examples of automated decision making in finite element analysis prove the feasibility and great practical potential of automated environments for design of aerospace structures. Based on studies performed in the project, directions of further research and development in this area were identified.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Spencer Wu, Ph.D.	22b. TELEPHONE (Include Area Code) (202) 767-6962
	22c. OFFICE SYMBOL AFOSR/NA

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

(w)

FINAL REPORT

NON–ALGORITHMIC ISSUES IN AUTOMATED COMPUTATIONAL MECHANICS

W. W. Tworzydlo, J. T. Oden, J. M. Bass, J. Combs, S. Sheikh

F49620
TR-89-C-0015

USAF, AFSC
AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
BOLLING AFB, DC

TR-91-09

April 30, 1991

THE COMPUTATIONAL MECHANICS CO., INC.
LAMAR CREST TOWERS
7701 NORTH LAMAR, SUITE 200
AUSTIN, TEXAS 78752
(512) 467-0618

COMCO

Contents

List of Figures

1 Introduction

The Finite Element Method (FEM) has been used in the analysis of structural designs for about two decades now. The field of computational mechanics has come to rely heavily on this technique, and gradually the FEM is becoming the most popular analysis procedure within various fields of design. This design and analysis procedure can be clearly divided into two types of processes. The first type involves performing large scale algorithmic computations and data processing. The second type of process involves decision–making which requires perception, intelligence, knowledge, and reasoning power.

This second type of process involving decision–making is traditionally performed by engineers who are expected to master the expertise necessary to effectively use the finite element software in the design process. Many computer aids like CAD software, graphic interfaces and on–screen data displays have helped to reduce the involvement of human designers by providing decision making data in an easily accessible form. Still, the involvement of a human expert in the decision–making process represents a major part of the time and effort of performing analysis and design for the following reasons:

- the scarcity of experts available to make correct and quick decisions
- the expense, in time and dollars, of training new experts
- the decision–making process itself, which is often routine and trivial, yet time–consuming

To overcome these problems, several research efforts are currently underway attempting to formalize decision–making criteria and to develop intelligent automated software to supplement the human designer. This report summarizes research performed in this direction at the Computational Mechanics Company, Inc.

1.1 Summary of Objectives of the Project

The primary goal of this project was to study the feasibility of automation of the decision–making process in both the mathematical methods (so–called deep knowledge) and artificial intelligence tasks (shallow knowledge) in the engineering design process. In particular, this study focused on automated numerical analysis and design by an advanced h-p adaptive version of the Finite Element Method. In this process, complex decisions are being made, to date primarily by the engineer, concerning:

- the mathematical model of the structure

- **representation of material properties**
- **selection of a computational method and strategy**
- **optimal mesh design**
- **optimal time step, load step, etc., selection**
- **handling of computational difficulties (divergence, zero pivots, etc.)**
- **verification of the reliability of finite element results**
- **modification of the model in order to satisfy design objectives**
- **optimization of the design**

Until recently, most of the above decisions were made by the designer and required considerable expertise in structural mechanics and numerical analysis.

The objective of this research effort was to study the feasibility of automating the decision making process in computational mechanics by means of novel algorithmic procedures (adaptive mesh refinement, adaptive time stepping), and by application of new computer technologies, designed to automatically handle nonalgorithmic decisions based on heuristics, experience, and human expertise.

1.2 Research Summary

The first year of the effort was devoted to assessing the current state-of-the-art in decision making software, evaluation of logical steps in the engineering design process, evaluation of possible tools (software) that can be used toward automation of this process, and the formulation of a general computational environment for coupling a finite element analysis with knowledge–based systems. Simultaneously, continuous progress on the theoretical basis for complex decisions involved in *h-p* adaptive finite elements was being made.

In the second year of the project the effort was focused on a detailed formulation of criteria and methods of automation of selected aspects of computational mechanics. Moreover, the concepts formulated in the project were verified by practical application of knowledge engineering software (Expert Systems) in the *h-p* adaptive finite element analysis. This effort, which was beyond the original statement of work, has proven to provide a great gain in efficiency, reliability, and ease of use that can be achieved by coupling knowledge engineering with classical methods of computational mechanics.

The particular tasks completed in the project are listed below:

1. An extensive literature survey was undertaken to assess the state-of-the-art in automated decision making, in particular with respect to applications in the engineering design process. The study indicated that some introductory efforts had been performed toward automating the selection of a structural model and computational strategies. Prior to this project, however, no complex integrated effort has been presented which automates the whole design process. In particular, no efforts have been reported on interactive coupling of finite element programs and knowledge–based systems.

2. The general, logical structure of the engineering design process was examined. The basic objective was to identify consecutive logical stages of this process as well as decisions made and knowledge used and generated at each stage. Possible methods and software necessary to automate consecutive stages were identified. The general conclusion was that full automation requires a synchronized process in both algorithmic and heuristic procedures.

3. An evaluation of the types of knowledge used in the engineering design process was performed. This includes, in particular, "deep" and "shallow" knowledge.

4. Evaluation of possible computer tools and software for handling deep and shallow knowledge in the engineering design of aerospace structures was performed. This includes algorithmic procedures, expert systems, object–oriented programming, knowledge acquisition, and neural networks.

5. A detailed study and development of methods, concepts, and algorithms related to the automation of the design process was performed. The issues addressed included:

   - selection of a mathematical model for given situations
   - selection of computational methods and strategies
   - design of optimal discretizations
   - development of adaptive computational methods for maximum effectiveness and robustness
   - verification of acceptability of final results

6. Further progress toward an automatic mesh refinement procedure in the adaptive h-p finite element method was achieved. Moreover, a procedure for the solution of nonlinear problems, with large deformations and contact constraints, was formulated and implemented. This procedure was automated by the application of knowledge–based expert systems.

7. A general computational environment for the interactive coupling of finite element programs and knowledge–based expert systems was formulated. This formulation includes the specification of functional elements of the integrated system as well as general data types and the data structure involved.

8. An extensive search and evaluation was performed on artificial intelligence software available today, its capabilities and the direction of evolution. This study indicates that simple rule–based expert systems are being replaced with much more powerful systems with object–oriented capabilities and even knowledge acquisition capabilities.

9. A direct coupling of the *h-p* adaptive finite element code with the Expert System software "Nexpert Object" was implemented. This implementation enables direct communication between the two technologies, so that heuristic decisions in the finite element analysis can be automatically handled by the knowledge engineering software.

10. Several advisory expert systems were implemented in the above coupled environment to automatically handle selected decisions in the finite element analysis. These include:

    - automated selection of computational strategies
    - monitoring and control of performance of the finite element computations
    - automatic verification of the mathematical model and finite element results

11. Representative examples illustrating performance of the coupled finite element–expert system environment were solved.

12. Based on the results of the project, further possibilities of automation of the design of aerospace structures were identified.

1.3 Personnel

The research effort was performed during the course of this project was provided by a highly specialized team of COMCO researchers. In particular, the Principal Investigator in the first year of the project was Dr. Jon Bass, Vice President of Science and Engineering at COMCO. In the second year of the project the Principal Investigator was Dr. W. Wojtek Tworzydlo, Senior Research Engineer and Manager of Advanced Projects Group I. Professor J. T. Oden, President and Senior Scientist of COMCO, was project supervisor. Assisting at different stages of the project were the following Graduate Research Engineers: Chris Berry, Olivier Hardy, Shakhil Sheikh, Tim Westermann, and Shibu Vadaketh.

During the course of the project, considerable help in issues related to object–oriented programming and knowledge engineering was obtained from a consultant, Jackie Combs, a specialist at the Artificial Intelligence Center at Lockheed.

1.4 Presentations and Publications

The research performed in the project was presented at the 8th Annual AFOSR Forum on Space Structures, held in Indialantic, Florida, June 18–20, 1990.

Certains aspects of this effort, namely interactive coupling of the adaptive finite element code PHLEX with the expert system software NEXPERT OBJECT, were presented in the form of a press release at AUTOFACT 1990.

Currently a paper summarizing the effort and entitled "Toward an Automated Environment in Computational Mechanics" is being prepared for publication.

1.5 Outline of the Report

This report presents the results of the literature survey, theoretical study, and practical design and implementation of an automated environment for engineering design by the finite element method. The report is divided into several sections discussing various aspects of the effort.

The first few sections present results of the literature survey and a theoretical study of issues relating to automated decision making in computational mechanics. In particular, Section 2 presents an updated survey of literature related to the project. Section 3 then provides a detailed analysis of the engineering design process with particular emphasis on choices and decisions made at consecutive stages of the design. In Section 4 different types of knowledge used in the engineering design are identified. This analysis is the basis for the evaluation of computer tools (languages and hardware) that can be effective in representing and using these types of knowledge (Section 5). In Section 6 is a more detailed discussion of methods, concepts, and algorithms for automated computational mechanics. This includes established numerical techniques, novel theories and algorithms as well as heuristic rules and facts.

The studies presented in Sections 2 through 6 are the basis for the layout and design of the general computational environment for automated design of aerospace structures by the adaptive finite element method. In particular, a general outline of the various components (pieces of software) of this environment and the general data structure are discussed in Section 7. In Section 8 a practical design and implementation of selected object-oriented expert systems is presented in detail. Several numerical examples are provided which prove the feasibility and illustrate the effectiveness of concepts developed in the project. This section is followed by conclusions and a list of references.

Parallel with the above studies a continuous effort is underway at COMCO toward optimizing the *h-p* adaptive finite element method. The basic formulation of this method and

an algorithm for tailoring it to the solution of nonlinear problems is presented in Appendices A and B. Detailed listings of knowledge bases implemented and tested in this project are compiled in Appendix C.

2 Literature Survey

As the first task in this project, an extensive literature survey was conducted to create a comprehensive background concerning the current state-of-the-art in various fields of knowledge which form a part of, or are a supplement to, the decision-making area of computational mechanics. It was noted that there are three major stages in the process of design and analysis in which extensive decision-making is required. These stages of decision-making are, in fact, responsible for most of the human interaction necessary during the entire process. The first stage is the modeling stage. At this stage the physical design has to be transformed into a model which the computer can understand and on which numerical analysis can be performed. Since most of the physical systems cannot be taken directly as the model on which the analysis is to be performed, several decisions are required so that the mathematical and numerical models represent the actual system as accurately as possible. This physical model must then be input in the form of a finite element mesh which is supplied or generated by the user. This task often requires a great deal of experience and knowledge.

The second major task requiring expertise is the numerical analysis of the model. This involves several complex decisions, in particular mesh design, selection of computational strategy, and choice of corresponding parameters, such as time step, load step, etc. It also involves decisions made when computational difficulties are encountered, for example, divergence of iterative method, unstable behavior of the solution, etc. Decisions made at this stage require both a solid theoretical background and considerable computational experience.

The third major task involving complex decisions is analyzing the results of the numerical analysis. At this stage, several decisions, such as acceptance or rejection of the design, must be made. If flaws are found in the design, remedial measures should be suggested and appropriate changes made to the initial design before the analysis process can be repeated.

Design and analysis is an iterative process that may require several iterations to achieve a final product. During some of the iterations the changes required are trivial, while others may require a great deal of expertise. To achieve an automated design process, it is necessary to automate each of the three stages mentioned above.

It is of importance to note that there are two basic types of knowledge and decisions made in the design process:

- algorithmic knowledge and corresponding procedures

- heuristic knowledge and decisions

A detailed discussion of these two types of knowledge is presented in Section 4. Here it is important to note that in the engineering design process there is a very strong interaction between algorithmic and heuristic knowledge and that in the automated environment they cannot be treated separately. Moreover, due to the continuous progress in science, some heuristic decisions become precise enough to be treated algorithmically, usually with better reliability, efficiency, and with the possibility of full automation. Thus, our literature survey has focused on the most advanced algorithmic methods useful in the automation of the design process as well as on the heuristic knowledge and software designed to handle it automatically.

Currently the major area in which algorithmic methods and decisions replace a heuristic approach and promise full automation is the adaptive finite element method or—more generally—adaptive computational methods. The term "adaptive computational methods" has become increasingly familiar in the modern computational mechanics literature as more analysts and engineers realize the great potential of the concepts underlying these methods. Adaptive methods, which are numerical schemes which automatically adjust themselves to improve solutions, include a wide variety of techniques. The more important of these are adaptive mesh refinement, adaptive adjustment of time steps in transient problems, adaptive load stepping in quasistatic nonlinear problems, or adaptive selection of implicit and explicit zones in transient analysis. An extensive survey on adaptive computational methods was recently compiled by Oden and Demkowicz [52] and Noor and Babuška [51]. The reader is directed to these works for detailed information.

The first work on adaptive finite element methods was presented in 1971 by Oliveira and Arantes [58] which discussed grid optimization by minimizing the energy by optimal node distribution. This type of approach—node redistribution—is the basis for the moving mesh adaptive methods (r-methods) developed for both solid mechanics problems and flow analysis [26,38,40,48,49,50,81].

Other adaptive finite element methods currently in use include:

- h-refinement based on a local refinement of the mesh without changing the order of interpolation [27,52,54,55]
- p-enrichment, in which the order of interpolation is increased locally to improve accuracy [25,71,72,73,83]
- h-p-method, in which both the mesh and the order of interpolation are adapted to minimize the error [4,29,30,53,61]

The most advanced method—and the most difficult to apply—is the *h-p* adaptive method. The advantage of this approach is that, while the conventional FEM's can provide only algebraic rates of convergence, an adaptive *h-p* method can result in exponential rates of convergence. It should be noted that the selection of *h*-refinement or *p*-enrichment is a difficult issue and the selection of an optimal sequence of refinements is still under development. A significant amount of progress in this direction was recently made by researchers at the Computational Mechanics Company, Inc., see references [29,30,53,61].

An alternative to a purely algorithmic approach to grid adaption is the use of heuristic knowledge to select *h*-refinement or *p*-enrichment. In this spirit, an expert system-like approach was developed by Babuška and Rank [6]. It is expected that the best results will be obtained from a combination of algorithmic and heuristic artificial intelligence approaches.

Adaptive mesh refinement is only one example of automated decision-making in the process of finite element analysis. Another important adaptive procedure is adaptive time step selection in the solution of time-dependent problems. The selection of a time step is usually based on deep knowledge, namely precise error estimators. Such estimators, based on the Courant-Friedrichs-Levy number, are extensively used in the solution of flow problems [54]. Similar procedures in solid mechanics, based on a truncation error analysis, were applied by Kumar, Majoria and Mukhurjee [41], Bass and Oden [10], and more recently by Thornton, Oden, Tworzydlo and Youn [74].

Yet another approach to adaptive computational methods or smart algorithms is represented by imlicit/explicit methods based on adaptive decomposition of the computational domain into implicit and explicit zones in order to maximize the efficiency and reliability of the computations. Such methods were recently developed for computational fluid dynamics problems by Tworzydlo, Oden, and Thornton [79].

The second major direction of research in the area of automated computational mechanics is the application of artificial intelligence to resolve decision-making problems. The field of artificial intelligence is, of course, not new. It has, however, only been recently that the developments in computer architecture have made the tools of artificial intelligence more practical, more widely available, and more user friendly. During the current decade, the idea of Knowledge-Based Expert System (KBES) has developed from mere theory to practical and integrated complex decision-making systems. Today there are numerous private companies and government agencies utilizing expert systems to solve decision-making problems in various fields of research, business, and defense.

It was in 1978 that the first steps were taken by Bennett, Creary, Englemore, and Melosh [12] at Stanford University toward forming an expert system for engineering design. The expert system was called "SACON" (Structural Analysis CONsultant) and could make intelligent decisions regarding forming the input model for a large finite element analysis program

called MARC. MARC was a flexible program capable of handling various types of analysis techniques, material properties and geometries. SACON could make decisions about the input parameters required by MARC for any specific design. This decision-making capability was generally mastered by an average engineer after working with MARC for about a year. Thus, SACON acted as an intelligent front end to the MARC program and advised inexperienced users about the best modeling approaches and the various parameter values required by MARC as input. The SACON had a knowledge base of about 170 IF,THEN rules which was controlled by the backward chaining EMYCIN inference engine.

SACON was a prototype and an operational version was never developed. The concept of expert systems initially did not seem feasible as the development and running cost were not economical and there were few computers at the time on which a program like SACON could work. In the 1980's, as the microcomputer and computer work stations developed and became more easily available, the idea of expert systems was reconsidered. At the same time, research began focusing on automating several processes which required decision-making. In the field of computational mechanics, the first area to be studied in this regard was automatic mesh generation. In order to exploit this new technology, it was required that expert systems or other AI tools be able to interact directly with existing CAD software and transform the generated models into a coarse mesh compatible with the FEM program. The various techniques of computer based geometric/graphical modeling currently in use are:

- wireframe modeling
- surface modeling
- solid modeling

The only technique out of these which can provide sufficient geometric knowledge about the physical model to the mesh generator, is solid modeling. Information available from wireframe models or surface models is not complete enough to be useful for fully automated mesh generators. Even solid models are limited due to the fact that feature description, such as that of holes, cavities, and other such discontinuities, is not sufficiently elaborate, even in the most sophisticated CAD programs. This causes CAD programs to fall short of independently providing the desired input for fully automated mesh generators. It is here that the expert systems or other AI tools are required to fill the knowledge gap.

In 1985 Fenves [32] (Carnegie-Mellon University) outlined a framework for what he called a knowledge-based finite element analysis assistant. He has illustrated the need for the intelligent interface of CAD and finite element analysis programs so that the available human expertise can be combined in a single software package and therefore reduce the time necessary for mastering the full design process. Fenves also concluded that the application of knowledge-engineering in this area has become a necessity.

In 1986 Gregory and Shepard [34] (Rensselaer Polytechnic Institute) developed a knowledge–based system known as FACS (Flexible Automated Conversion System). FACS had the capability of interacting with various CAD programs and FEA solvers through specific translators. It could take the geometric specifications of various airframe components provided by the CAD program and was able to generate from this data a coarse mesh to be input to the finite element solver.

Cagan and Genberg [19] (University of California, Berkeley, 1987) also developed an expert system in this area. The expert system was called PLASHTRAN (PLates And SHells STRuctural ANalysis) and acted as an advisor and learning aid to the users of a large finite element analysis package called NASTRAN. For ease of development, only two–dimensional elements were considered. The knowledge framework was created in an object–oriented environment known as LOOPS (Lisp Object–Oriented Programming System). The inference engine used was a data–driven forward chaining type, allowing the expert system to follow the logic of the user rather than the user having to follow the logic of the shell, as it is for the more rigid systems based on strictly backward–chaining shells.

Blacker, Mitchiner, Phillips and Lin [14] (Sandia National Laboratories) have also produced an automated two–dimensional quadrilateral mesh generator using a transfinite mapping algorithm for generation of the elements. It is based on a set of higher–order primitive decomposition algorithms. The decision–making is integrated into the process by using a Knowledge–Based System built around Common LISP and the KEE (Knowledge Engineering Environment) shell. The program has thus been named AMEKS (Automated MEshing Knowledge System). AMEKS uses a recursive strategy to successively remove meshable primitives, exhaustively decomposing the initial region. Likely dissections are attempted until an acceptable primitive is produced. The strategy is repeated on the remaining regions. The decomposition is completed when a dissection results in two acceptable primitives. The mesh is then algorithmically produced for the decomposed regions. This program can handle both straight and curved two–dimensional geometries but cannot handle subfeatures like holes or other such discontinuities.

The trend of applying tools of artificial intelligence to automate various stages of decision–making in engineering design has gained further momentum and several research efforts have been made in this area. Some of these are mentioned here and several others are listed in a comprehensive bibliography.

The work of Chen and Hajela [22] (1989) has produced the expert system OPSYN (OPtimum SYNthesis). This system has the capabilities of finite element modeling, optimum design modeling and selection and calculation of optimization strategies. The knowledge base provided for finite element modeling contains rules for selection of node location and numbering and for generating the initial mesh. Rules pertaining to mesh refinement, element

selection and element distortion elimination have been developed as well. The knowledge is represented in the form of IF, THEN rules and its application is controlled by an inference engine (INFER) capable of both forward and backward chaining. OPSYN goes a step further than most other expert systems by implementing a knowledge acquisition mode. In this mode, specific problems are presented to experts to solve interactively. The method of solution or the decision made by the experts are stored in a file in the form of IF, THEN statements with confidence levels specified. These can then be used by the knowledge engineer to formulate new rules.

An-Nashif and Powell [3] (University of California, Berkeley, 1988-89) have also worked in this area of research in conjunction with the automated modeling of frame structures. They use a system of "components" and "connections" for defining the basic structure. This structure is then transformed into the analysis model which is defined in terms of nodes, elements, and boundary conditions. The strategy defined here has yet to be implemented in the form of a working knowledge-based system, but it seems to lend itself quite favorably to object-oriented programming.

Sapidis and Perucchio [66] (University of Rochester, 1989) have worked in the area of producing finite element meshes automatically from solid models. They have analyzed and evaluated the algorithms dealing with element extraction, domain triangularization, and recursive spatial decomposition. A review of some existing expert systems, which integrate the decision-making knowledge with these algorithms to produce various levels of automation in mesh generation, is also carried out. The authors conclude that, although significant progress has been made in the field of automatic mesh generation, no system exists which can be considered fully automated.

Another approach to the decision-making problem has been established by Ohsuga [57] (University of Tokyo, 1989). He presents the idea that the CAD programs themselves should be made "intelligent" so that they can perform tasks other than that of pure drafting. Ohsuga argues that the current CAD programs lack the sophistication, flexibility, and innovation required for automated design synthesis. He also argues that the traditional artificial intelligence tools, like rule-based expert systems or object-oriented programming alone, are also insufficient to handle the intricacies of automated design synthesis. As a solution to these problems, he presents KAUS (Knowledge Acquisition and Utilization System), an intelligent CAD environment based on a knowledge representation language developed especially for this purpose. KAUS, though not yet fully developed, still shows great flexibility in the design synthesis [44] process and may prove to be one of the forerunners in the intelligent CAD environments of the future.

From the study presented above one can conclude that practically all reported efforts toward automation of the design process have focused on the first stage of design, namely

understanding of the physical problem and construction of the numerical model, in particular the finite element mesh. No successful efforts were reported of the application of expert systems at further stages of the design, in particular in the control of the performance of the finite element procedure or in the analysis of numerical results and possible redesign of the analysis model.

Another general observation is that most of the systems or concepts developed so far were implemented as stand–alone advisory pieces of software, not integrated with other elements of the design environment. One of the reasons for this situation was that until recently most of the knowledge engineering applications were developed using the LISP language, which makes coupling of expert systems with computer languages used in numerical analyses (FORTRAN, PASCAL, C or others) extremely difficult.

Further difficulties in practical applications of the systems reported so far is due to the extremely arduous and complex development and management of a realistic knowledge base, with the number of rules exceeding several hundred. Losing control over the size of the knowledge base is a trap which can ensnare even the most experienced experts if caution is not exercised. Such efforts can easily cause the spectre of conflicting rules and uncontrolled linking of knowledge to haunt the developer. A possible solution to this problem is specialization by development of smaller, specialized expert systems or segmentation of larger knowledge–based systems and encapsulation of knowledge. This approach was actually used in the present work.

The general conclusion of this survey is that the concepts of tools for making automated decisions in the design process are presently emerging in the engineering community. While still very limited in the degree of automation, generality and realistic capabilities, these efforts indicate that the need for such systems exists and that intelligent design environments will be developed in the future.

It should be noted that a fully automated system which can take the user completely out of the loop (especially for a complex and conceptual task such as design synthesis), is still far from reality and a thing of the future. Even if this is made possible by advances in fields such as neural networks (or other such tools which may be devised in the years to come), the question will remain: can a machine ever be as innovative as the human mind?

3 Decision Making and Knowledge Flow in the Engineering Design Process

The process of engineering design is oriented toward the construction of a physical system to perform specified functions. The complete design process involves considerable technical

effort as well as the analysis of cost, timing, manufacturing, etc. In this presentation we will focus primarily on the technical aspect of the design process. The technical objective of design is to produce a model of a product which will perform specified functions under certain conditions, usually for a specified time.

The general overview of consecutive logical stages of the design process is presented in Fig. 3.1. The process begins with the identification of physical problems and design objectives and—usually after several modifications—yields a model of a product performing specific functions and satisfying design criteria.

During the design process, knowledge about the model is enriched, modified, and used at further stages of the design. In Fig. 3.1, the total knowledge accumulated during the process is stored in the common data base, called the blackboard (this name corresponds to the generic way of handling knowledge, known as blackboarding). This idealization does not necessarily mean that blackboarding is the only or the best way of storing knowledge in practical applications. The various components of the chart of Fig. 3.1 use, modify, and constantly update the information stored on the blackboard. In the following sections, consecutive stages of the design process will be briefly discussed and evaluated.

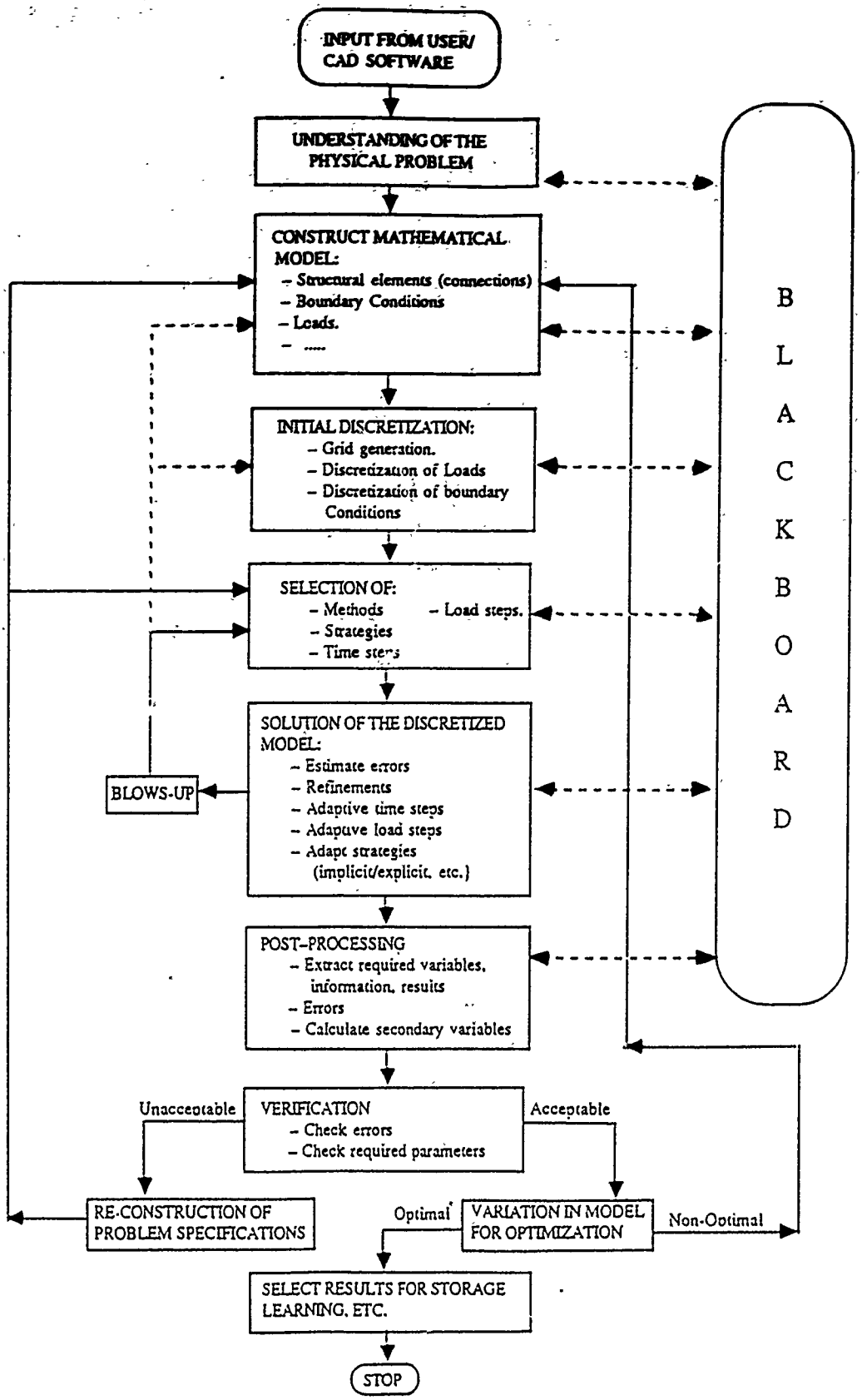
The primary goal of this analysis is to identify the decisions made at different stages of the design process, the knowledge base used to make these decisions, and the criteria used to make correct choices. For each of the stages of analysis, we try to identify possible tools (types of software) that can be used toward automation of the design.

3.1 Understanding of a Physical Problem

At this introductory stage of design, all distinctive features of a structure are identified and the principal design objectives are specified. The structure is logically decomposed into simpler units and their basic mechanical features are identified. The general class of loads is recognized and the importance of possible environmental, chemical, electromagnetic, and other influences is assessed. Different design objectives are specified, such as the functions to be performed by the structure, required life time, desired reliability under random loads, and interaction with other structures.

This stage of design is the basis for the selection of the mathematical model of the structure. As a result of this introductory analysis, decisions are made as to whether a simplified form of the governing equations is to be applied or whether a more complete set of equations is required.

The decisions made at this stage of the analysis are mostly heuristic, based on the designer's expertise or previous experience. Until recently, this stage of design was performed solely by a designer or group of experts. Recently attempts have been made toward devel-


INPUT FROM USER/ CAD SOFTWARE
UNDERSTANDING OF THE PHYSICAL PROBLEM
CONSTRUCT MATHEMATICAL MODEL:
– Structural elements (connections)
– Boundary Conditions
– Loads.
– .....
INITIAL DISCRETIZATION:
– Grid generation.
– Discretization of Loads
– Discretization of boundary Conditions
SELECTION OF:
– Methods
– Strategies
– Time steps
– Load steps.
SOLUTION OF THE DISCRETIZED MODEL:
– Estimate errors
– Refinements
– Adaptive time steps
– Adaptive load steps
– Adapt strategies (implicit/explicit, etc.)
BLOWS-UP
POST-PROCESSING
– Extract required variables, information, results
– Errors
– Calculate secondary variables
Unacceptable
VERIFICATION
– Check errors
– Check required parameters
Acceptable
RE-CONSTRUCTION OF PROBLEM SPECIFICATIONS
Optimal
VARIATION IN MODEL FOR OPTIMIZATION
Non-Optimal
SELECT RESULTS FOR STORAGE LEARNING, ETC.
STOP
BLACKBOARD

oping expert systems supporting the user at this stage (see the literature survey in Section 2). Specialized expert systems for specific applications can be developed, such as an expert system for the structural design of airplane wings. These systems can be grouped and interactively used for the introductory evaluation of complex structures.

The general knowledge of the system, accumulated at this stage, is the basis for the formulation of geometrical and mathematical models at further stages of the analysis.

3.2 Mathematical Model of the Structure

Once the essential features of a physical model have been established, the mathematical model of the structure is constructed. Formally, this mean that a boundary value problem or initial boundary problem is formulated in a certain domain. Practically, it means that the equations are selected to represent the geometry and deformation of the structure, the properties of the material, boundary conditions, loads, constraints (such as contact with other structural members), and other effects. This stage involves two functionally different actions:

- definition of the domain Ω, i.e., the geometric shape of the structure,
- selection of the mathematical formulation which best represents the physical behavior of the structure.

The above distinction is important from the application point of view, because significantly different tools will be used to aid the designer at these tasks. For the definition of the shape of the structure, a CAD–type solid modeling software is probably best suited. The progress in this area is advancing rapidly and there is a large variety of CAD programs currently available, and most are equipped with excellent user interfaces and graphic capabilities.

The selection of the mathematical model is more difficult and requires considerable theoretical background. The choice of the actual model depends on several factors, some of them highly quantifiable, others somewhat heuristic. A more detailed discussion of these issues and a presentation of possible systematic approaches to the selection of a mathematical model are presented in Section 6.1. Here we only note that the practical automatic selection of a mathematical model requires application of a powerful expert system, capable of combining heuristic information with analytical evaluations according to the theoretical estimates.

3.3 Discretization of the Model

Most computational methods used today, particularly the finite element method, are based on an approximation of the original problem by a problem formulated in the finite–dimensional space. In geometrical terms this requires discretization of the domain into a mesh of nodes and elements, as well as discretization of boundary conditions and loads. These tasks were traditionally performed by the users of finite element codes. Considerable expertise was required to design a proper mesh, in particular, to concentrate nodes in expected areas of high gradients of the solution. Recently, several CAD programs have emerged that automatically perform discretization of the domain with the use of an automatic mesh generator. Still, in such a system, the designer must specify the overall characteristics and density of a finite element mesh.

As indicated in the literature survey (Section 2), several recent research efforts were directed toward application of knowledge–based expert systems to automate the discretization process. Although these efforts are certainly of great importance, it is our belief that the application of adaptive computational methods will make one of the most complex components of such an expert system obsolete. With the application of adaptive mesh refinement, there is no need for the user to anticipate the nature of final solution or to concentrate nodes in certain subdomains. The initial mesh should merely represent the basic geometry of the domain, and the adaptive mesh refinement procedure, based on sharp error estimates, will yield a final mesh more adequate to the problem than even the most experienced expert could generate.

In conclusion, the automated generation of an initial discretized model will involve the application of CAD software, combined with a knowledge–based system and—for flexibility—a graphic user interface.

3.4 Selection of Computational Methods and Strategies

The general purpose finite element codes available today offer a wide spectrum of computational strategies for the solution of various problems. For a given problem, the user must select a specific method, as well as the size of the time step in time–dependent problems, the size of the load step in nonlinear problems, etc. This selection is usually based on general guidelines and the previous experience of the user. Unfortunately, in most areas of computational mechanics, there exists no unique, universal, or absolutely best computational strategy. For example, in computational fluid dynamics, there is a variety of finite element algorithms—Taylor-Galerkin, Runge-Kutta, SUPG, least squares—that are all adequate and yet none has proven to be generally superior to the others. Thus, proper selection of the algorithm for a specific problem can yield better results, often with less computational ef-

fort. This is especially important in nonlinear or time–dependent problems, because it often means the difference between a convergent or divergent process. In this case, the presence of an experienced user is crucial in the applications of today's computational systems.

Apparently, the automatic selection of a proper computational strategy can readily be achieved with the application of expert systems. Indeed, some introductory attempts toward building such systems were already presented in the literature (see Section 2). It should be noted that the systems that have been developed (for example SACON [12]) are stand-alone programs, requiring considerable input from the user. In an integrated approach, as presented in Fig. 3.1, most of the information used at this stage should, instead, be drawn from the knowledge base developed at earlier stages of the design process.

There is, however, one additional important aspect of the automatic selection of a computational strategy. As all engineers familiar with nonlinear problems know, it takes tens or even hundreds of solved problems—both convergent and divergent—to develop a certain "feel" for the proper selection of a computational strategy, load sequence, size of the load step, etc. Therefore, it would be extremely beneficial to use, at this stage, software with knowledge acquisition possibilities. Such a system could automatically gain experience from previous applications and use it in future analyses of problems with similar characteristics.

3.5 Numerical Analysis of the Discretized Model

The discrete model of the structure being designed is usually analyzed by the finite element method (or any other numerical method). This stage of the design process usually amounts to massive algebraic computations. In the majority of finite element codes, it is assumed that all decisions and choices have already been made and the program follows a prescribed procedure to produce final results or, quite frequently, give error messages or just stop execution. This approach often leads, especially in the case of advanced nonlinear applications, to considerable problems for the user and wasted computational effort because:

- if an error occurs, there is usually no suggestion as to how to fix it,
- if the nonlinear process or any iterative procedure diverges, there is neither an explanation to the user about possible reasons, nor suggestions as to how to modify the data (or change the strategy) in order to achieve convergence,
- if the time step in a time–dependent process is too large, there is no indication of a large error (or even instability) provided to the user,
- after the final results are obtained, there is no aid provided for the user to estimate the reliability of the results, the relevance of the user–defined mesh, etc.,

- often a full cycle of expensive computational needs to be executed just to learn that the results are not acceptable and the data needs to be modified.

The general conclusion is that with today's finite element codes the user has very little guidance as to ways of handling errors, divergence, instability, or how to assess the quality of the results produced.

The solution to this problem is the automation of the finite element analysis by the application of adaptive computational procedures as well as utilizing the knowledge–based expert systems to automatically handle computational difficulties. Thus, adaptive computational methods are very attractive in this context, because they offer the possibility of:

- automatically making decisions or choices previously made by the user and requiring a considerable level of expertise,
- minimizing the computational effort necessary to obtain high quality results, and
- automatically controlling the quality of results by means of error analysis.

The specific types of adaptive computational techniques currently being developed for the finite element method include:

- adaptive mesh refinement
- adaptive selection of time steps in transient problems
- adaptive selection of load steps in nonlinear problems
- adaptive application of implicit/explicit algorithms in time–dependent problems

Adaptive methods present a very natural step toward the automatic design process because they inherently include automatic decision making in the process of numerical analysis. These decisions are, in general, both algorithmic, based on error estimates, as well as heuristic, based on general guidelines and experience. Therefore, a fully automated adaptive procedure will require coupling of the finite element analysis program and a knowledge–based intelligent system.

Although adaptive techniques can, in general, provide automatic control of the solution process, there exist situations, e.g., in the case of erroneous input data, that the errors, instabilities, divergence, or other computational difficulties do occur during the analysis. In the fully automated process such situations should be detected, signalled to the user, and some measures toward fixing the problem should be suggested. A typical example of such a situation is in the analysis of a strongly nonlinear problem, say the rolling of a tire, where

the convergence of a Newton procedure is not necessarily guaranteed and is sensitive to such parameters as selection of the load step, load sequence, penalty parameter, mesh distortion, etc.

An effective knowledge-based system should be capable of accumulating the knowledge of the author of the program or the experienced user and offer guidance to the inexperienced engineer or even take over the control of the program to achieve successful computations. In the most advanced situation, an intelligent system should automatically build its own experience based on successful and unsuccessful executions of the code.

3.6 Generalized Post–Processing

The final solution of the numerical simulation is often obtained in the form of tables of nodal values of the basic unknowns of the problem, e.g., displacements in the solid mechanics problems. This solution is usually post–processed in order to present the results in a more comprehensive form or to calculate new functions of interest to the user. Typical functions of post–processing modules are:

1. calculation of auxiliary functions (strains, stresses) derived from the primary solution,
2. use of special post–processing techniques to obtain results of higher accuracy than the primary solution,
3. estimation of solution errors, and
4. presentation of results in the form of plots, diagrams, tables, etc.

The above tasks are basically algorithmic. However, advanced programs usually offer several possibilities for performing these tasks and the selection of the method is expected from the user. This is especially true for items two and three, since a variety of algorithms, each of different quality and computational cost, is available for advanced post–processing or error estimation (see references [7,8,9,15,16,60,84]).

The post–processing stage is important in the design process for yet another reason: it provides information basic for the acceptance or rejection of the solution and for modification of the input data in order to obtain a satisfactory solution. New methods of postprocessing designed to specifically serve this purpose were designed in this project and are discussed in Section 6.3.

3.7 Verification of the Finite Element Solution and Modification of the Model

The results obtained after the first pass through the finite element solver are very seldom satisfactory. Usually modification of the data is required, either to obtain better quality of the solution or to modify the model in order to satisfy specific design criteria and objectives. While the satisfaction of general design objectives usually involves considerable creative thinking and requires human invention, some basic criteria can be readily verified by the knowledge-based system. In particular, an expert system can make sure that the error of the solution is within prescribed tolerance, that maximum stress does not exceed limits, that displacements are not too large, etc. Moreover, the system should be prepared to suggest possible solutions in the case of violation of certain design criteria or even take over the control and do the necessary modifications.

A special case of modification of the model is the situation, when the objective of the design process is optimization toward minimum weight, maximum strength, or some other goal. The basic approach to optimization is to perturb control variables so as to minimize the goal function, with the model still satisfying basic design criteria. A variety of algorithmic optimization techniques exist that can be applied at this stage. The knowledge-based expert system can be used at this stage for the selection of the optimization method, as well as interact with the algorithmic method in more complex cases.

3.8 Accumulation of Experience

The human designer gains new experience from every design or analysis performed. This experience is used in the next design task, especially if the features of the new problem resemble characteristics of problems solved before. This accumulated experience constitutes a designer's professional expertise and differentiates an experienced engineer from a novice. The professional experience is somewhat intuitive and rather difficult to transfer, especially since not all engineers are willing to share it "free." This remark pertains in particular to the effective use of a finite element program or other software.

In this context, it would be very useful to have the possibility of using specially designed software to automatically save the experience of each design session and use it in future applications. New emerging types of software with certain learning capabilities are discussed in Section 5. The two most promising kinds of such systems are:

- knowledge-acquiring expert systems
- neural networks

Note that even with the automatic learning capabilities available it will be difficult to specify a compact set of essential information about each solved problem to be saved as an "experience" for future use (saving all the static and active knowledge from each session would not be feasible). Apart from these difficulties, the practical payoff of learning capabilities will be enormous.

4 Types of Knowledge in the Engineering Design Process

Engineering design and analysis can be viewed as the process of building a more and more broad and precise knowledge about the structure being designed. The engineer begins the design with certain general ideas about the structure and a list of design objectives and criteria. During the design process engineering skills are applied to build and accumulate more precise knowledge about the system. The final product of the design process is the specific shape and material of the structure, as well as information concerning the possible distribution of stresses, plastic zones, and even an estimated number of load cycles the structure can sustain.

The knowledge used and accumulated in this design process can be divided into two classes:

- Static knowledge, representing all the information about the system analyzed, such as physical structure, material constants, corresponding finite element mesh, and, after solving the problem, displacements, stresses, discretization errors, etc.
- Active knowledge, including physical laws, mathematical equations, and heuristic rules used in the design process.

For automated decision making, the representation and handling of active knowledge is very important. Depending on the form and precision of the rules, the active knowledge can be divided into two groups:

- deep knowledge, represented in the form of precise physical laws, mathematical formulas, and computational algorithms,
- shallow knowledge in the form of heuristic rules pertaining to the problem and resulting from some general observations, experience, and even intuition.

It should be emphasized that this distinction is somewhat arbitrary and that the division line is never fixed. In particular, in the evolution process of science and technology one can

observe the transfer of some rules from the class of shallow knowledge to deep knowledge. For example, until recently the selection of a good mesh for the finite element method was an art, requiring considerable experience and intuition. However, the development of methods of adaptive mesh refinement, based on precise error estimates, replaces this intuitive approach and makes the selection of a mesh a robust, reliable and automatic process.

In today's computational systems the deep knowledge is usually represented by the formulae and algorithms coded in the program, while the designer is supposed to furnish heuristic, shallow knowledge and creative thinking. In the automated approach to computational mechanics not only the algorithmic part, but also the shallow knowledge should be handled by the program, so that the engineer can concentrate on creative thinking.

Toward this end, both aforementioned types of knowledge will be analyzed in the following sections. Particular attention will be paid to the possible numerical representation of each type of knowledge and effective interactions of both types in the engineering design process.

4.1 Deep Knowledge

Deep knowledge is—in the context of the design process—a knowledge that can be precisely specified in a closed form of mathematical formulas or computational algorithms. This knowledge is usually derived from objective laws of physics, mathematical theorems, or numerical analysis theory.

A typical example of deep knowledge is a constitutive relation for a linearly elastic material:

$$T = CE \tag{4.1}$$

where T is a stress tensor, E is a strain tensor, and C is a fourth order elasticity tensor. Another example of deep knowledge is the Jacobi iterative procedure for the linear system of equations:

$$Au = b \tag{4.2}$$

where u is the solution vector, A is the coefficient matrix, and b is the right–hand side vector. The basic operation of the Jacobi iterative procedure for this system is defined by:

$$u^{n+1} = D^{-1}(D - A)u^n + D^{-1}b \tag{4.3}$$

where u^n, u^{n+1} are consecutive guesses for the solution u, and D is the diagonal of the matrix A. A considerable amount of deep knowledge is implied in (4.3), including convergence and stability theorems, estimates on the number of operations, etc.

It is important to note that, although these examples of deep knowledge are indeed very precise, the selection of the piece of deep knowledge to be used is often heuristic,

based on shallow knowledge. For example, the decision to apply Hooke's constitutive law to describe the behavior of a material is often based on experience, general guidelines, and even engineering intuition. Similarly, there exists a variety of solvers for linear systems of equations and the selection of the solver for a given application is still based on general guidelines or even personal preference.

With the automation of the design process as the ultimate goal, it is important to note that deep knowledge is usually algorithmic and often amounts to massive numerical computations. The algorithmic languages such FORTRAN, PASCAL, or C, are believed to be suited best for these applications and are commonly used today.

The wide variety of deep knowledge used in the engineering design process includes all areas of technical science, in particular mathematics, structural mechanics, materials science, fluid mechanics, chemistry, numerical analysis, and many others. Most of this knowledge is well established, documented in books and scientific papers, and implemented in various engineering programs. There exists, however, a "frontier" in many of these areas, where new models, methods, and algorithms are being developed. The knowledge for these methods is often not yet complete, and is—in the present state—augmented or completed by heuristic information. Moreover, many "deep knowledge" methods feature conditional applicability, stability, and convergence, and thus require additional expert knowledge to be used effectively.

In the fully automated design environment, the application of these models and methods requires coupling of algorithmic deep knowledge and heuristic, expert type shallow knowledge. This is especially true in the area of computational methods, where new methods and techniques are developed continually.

The area of adaptive computational methods has been one field that has experienced substantial attention in recent years, particularly with regards to:

- adaptive mesh refinement
- adaptive selection of time steps in transient problems
- adaptive selection of load steps in nonlinear problems
- adaptive application of implicit/explicit algorithms in time-dependent analysis

Adaptive techniques are especially interesting in the context of the automation of the design process because:

- They greatly reduce the computational effort necessary to obtain final results and, at the same time, provide control of the quality of results.

- **They eliminate a number of decisions, which in traditional versions of finite element codes are expected from the user and require considerable theoretical background and experience. Therefore, adaptive techniques are, "by definition", oriented toward automation of the design process and automated decision making.**

- **They inherently include decision making in the process of numerical analysis. Although the ultimate goal for an adaptive package is a purely algorithmic approach, in the present state, a considerable amount of heuristic knowledge and decisions is required. It can be expected that even in the most advanced forms of these algorithms, heuristic knowledge provided by expert systems will be needed for the best performance.**

Practical application of adaptive methods with feedback from knowledge–based systems is one of the most challenging problems of both computational mechanics and artificial intelligence, since it requires interactive coupling of sophisticated algorithmic software and knowledge–based systems.

In the present project, the most advanced version of adaptive mesh refinement, namely, the *h-p* adaptive finite element method was used. This method, applying simultaneously *h* mesh refinement and polynomial enrichment, gives rates of convergence exceeding the performance of other simpler refinement techniques. Although the method is still under development, significant progress has been made by COMCO engineers toward a fully automated *h-p* version, with operational *h-p* data structures, error estimates and equation solvers. A brief overview of the *h-p* method is presented in Appendix A and more detailed information can be found in references [29,30,53,61].

In a fully automated environment for engineering design and numerical analysis, a new family of deep knowledge will be necessary. As it is well known, basic results produced by finite element or finite difference programs are in the form of long tables of numbers, representing nodal displacements, stresses, etc. No expert system, or, for that matter, human expert, can effectively examine these data and decide whether the results are acceptable or a modification of the initial model is necessary. Therefore, methods are required to extract *essential* information from the massive set of data produced by these codes. The simplest pieces of this essential information are, for example, maximum or effective stress, maximum strain, maximum deflection, etc. In addition, information concerning the distribution of stresses, localization of the highest stresses, range of plastic zones, etc., is also necessary. In programs oriented toward interaction with human experts, graphical presentation of the results is most commonly used. However, for an expert system interacting with a finite element code, this graphical information is unnecessary. Instead, a limited amount of data should be provided to represent essential information about the solution. This issue is addressed in Sections 6.3 and 6.4.

4.2 Shallow Knowledge

Decision making is a complex process which, until recently, was confined to the domain of human capabilities only. Two key factors in making any decision have imposed these restrictions:

- **domain knowledge**
- **thought process**

"Domain knowledge" is the knowledge and experience about the area in which the decision is to be made. This can consist of a list of pertinent facts, methods, and their resultant outcome. Since the majority of this knowledge cannot be presented in the form of precise mathematical statements and theorems, this type of knowledge is often identified by the name of "shallow knowledge" (as opposed to "deep" procedural knowledge).

"Thought process" consists of reasoning or, more generally, the application of available knowledge in an intelligent fashion to reach a feasible decision.

In the engineering design process the "shallow knowledge" and related decisions are used at virtually all stages: beginning from the understanding of the physical problem until the final acceptance of the devised model. If we restrict our attention to the stages associated with the numerical analysis, the particular decisions based primarily on the shallow knowledge include (at the current state of the engineering science):

- Selection of the mathematical model of the structure (e.g., beam versus plate, elasticity versus elastoplasticity, etc.).
- Construction of the initial finite element mesh.
- Selection of the computational strategy, for example, specific method time integration (e.g., backward difference versus Crank-Nicholson), or methods of solution of nonlinear problems, extraction of eigenvalues, etc.
- Handling of computational difficulties and errors, for example a divergent nonlinear iterative procedure or zero pivots in the linear system of equations.
- Acceptance or rejection of the finite element solution and of the final designed model.

The shallow knowledge is also used to some extent in rather algorithmic stages of the finite element analysis, for example:

- Adaptive mesh refinement (selection of error criteria, refinement/unrefinement in specific areas, etc.).

- **Adaptive implicit/explicit method or other zonal methods.**
- **Mathematical post–processing of finite element results, namely the calculation of higher–quality results (e.g., stresses) from the basic solution or the estimation of errors of the final solution.**

It should be noted that shallow knowledge is seldom used in the "pure" form. Usually it is associated with elements of deep knowledge. For example a change of the constitutive model from linearly elastic to elastoplastic is based on information (stresses, strains) obtained from algorithmic procedures. This means that a fully automated environment for engineering design should allow for a very close coupling and integration of the software handling algorithmic and heuristic knowledge. It is our belief that the lack of such coupling was one of the reasons for rather limited success and application of stand–alone advisory systems developed previously for engineering application (e.g., SACON, PLASHTRAN). Effective interactive coupling of finite element procedures with knowledge engineering is one of the major achievements of this effort.

5 Review of Computer Tools for Automated Computational Mechanics

Development of an automated environment for engineering design will require the combination of a variety of computer tools and the development of appropriate interfaces between them.

On a similar basis, as to the types of knowledge, these computer tools can be divided into two general groups:

- Procedural software, designed to effectively implement deep, algorithmic knowledge, and
- "intelligent" software, designed to manage shallow, heuristic knowledge.

For decades these two major groups have been developing separately with very little interaction. As a result, the methodologies, computer languages, and even hardware used in these groups are very different. However, in recent years successful efforts have been made to narrow the gap between the two groups and to enable a closer coupling of various pieces of software. Such a procedure will be developed in this project. Before discussing details of this procedure, we will briefly review the major characteristics of computer tools belonging to the two aforementioned groups.

5.1 Algorithmic Procedures and Data Processing

Processing deep, procedural knowledge usually amounts to massive algebraic computations following the user–selected algorithm. Typical examples of such applications include:

- CAD design
- mesh generation
- engineering analysis by the finite element method or other methods
- post–processing of the results (including graphic presentation)

There exists a variety of software programs available for each of these applications, in particular for the finite element analysis. (A detailed presentation of this software is beyond the scope of this report.) In this project a finite element code, based on the most advanced *h-p* adaptive version of the finite element method, was used. This code is a typical representative of a large group of engineering analysis codes, in the sense that:

- It was developed in the FORTRAN language, traditionally used for large–scale algorithmic operations for engineering applications.
- It is devised to run on computers with relatively large computational power, from advanced workstations to supercomputers (however, even PC versions with limited capabilities are available).

Recently there exists a tendency, in particular in the development of pre– and post–processors with graphic capabilities, to use other, more powerful and versatile languages, in particular C. This is also a language usually used in the development of CAD or solid modeling software.

5.2 Object–Oriented Programming

Object–oriented programming (OOP) [47] methodology was considered in this project for several reasons. In addition to the general benefits of code modularity and reuse that OOP provides, OOP capabilities also enable us to represent the data in a format that is amenable to both algorithmic computations and heuristic reasoning. We are also able to use the OOP data to model a particular engineering problem at different levels of abstraction (e.g., the engineer interacts with the data at the physical object level, whereas the finite element model is composed of the very small numerical of the objects).

In computer systems developed using traditional procedural languages the focus of the system's analysis, design, and development is on the functionality of the project. In OOP the focus is on classes and objects. One would describe a traditional procedural program as a set of functions applied to arguments:

$$f(xy)$$
$$g(x)$$

In OOP the emphasis is on the objects:

Send the object x the message to perform operation g.

Even in rule–based systems where the basis of the conceptual model is on IF THEN rules, when the rules are reasoning over objects, the focus of attention in the analysis, design, and development of the rule–based system is still on the objects. In other words, for all things object–oriented, the conceptual framework of the system is the object model.

The four major elements in the object model are: abstraction, encapsulation, modularity, and hierarchy. These elements provide the expressive power needed to define the model of a complex real–world problem.

Abstraction is one of the fundamental ways human beings cope with complexity. In this way we are able to capture the core characteristics of an object that distinguish it from other objects. The set of core characteristics of an object describes its properties and behavior. An abstraction captures the outside view of an object, and it serves to separate the object model from its implementation.

The concept of *encapsulation* is complementary to the concept of abstraction. Whereas abstraction places the focus on the overview (outside view) of an object, encapsulation enables the developer to place a wall around the code that implements the object and at the same time provide access to information about the current state and behavior of the object. One can think of an object class as having two parts: an interface and an implementation. The interface is equivalent to a class's outside view. It consists of the abstract behavior common to instances of the class. The implementation is equivalent to a class's internal view and comprises the data format used to maintain the current state information of the object as well as the mechanisms that exhibit the behavior of the object. Encapsulation allows the developer to hide from the user all of the details of an object and at the same time provide user–access to all the essential characteristics of the object.

Modularization is the act of decomposing a complex problem into individual, manageable components. Partitioning a large complex program creates a number of well–defined, documented boundaries within the program. This enables us to view the program from various levels of abstraction. At the highest level the model may be a set of objects with each object representing a module. At a lower level of abstraction the module itself is the model consisting of a hierarchy of classes and their objects. Modularity not only assists in

the development of a system, it impacts the entire system life cycle. A system that has been decomposed into a set of cohesive and loosely coupled modules is more amenable to extension and maintenance.

The three object model elements previously described—abstraction, encapsulation, and modularity—all assist in reducing the complexity of a large system, but this is still not enough. By identifying that a set of abstractions forms a *hierarchy* we greatly simplify the model. The two most important hierarchies in a complex system are its classification (kind of) structure and its assembly (part of) structure.

The classification structure provides the inheritance capability, an essential element in object-oriented systems. Basically, inheritance defines the relationship that exists when one class shares the structure and behavior of one or more other classes. A subclass usually extends or redefines the structure and behavior of its superclasses.

Whereas classification structures denote generalization and specialization relationships, assembly structures describe aggregation relationships. For example, a car is made up of a collection of sub-objects: a motor, wheels, body, suspension system, etc. These assembly hierarchies can be viewed as different levels of abstraction, which is described earlier in this report.

These two types of structures form the basis for two important properties of OOP: reuse and object focus. Inheritance provides for reuse (e.g., the operation to compute the speed of an object in a two-dimensional plane needs to be defined at the most general object class) and object focus (e.g., one can reason about all instances of the class SHIP in the domain, or focus on the instances of the subclass SUBMARINE). In much the same way the assembly structure allows for reuse (e.g., the code for describing the current state of a WHEEL that is part of a CAR assembly could be the same as for the WHEEL of an AIRPLANE assembly) and object focus (e.g., different interfaces to the object in question could view it as a single entity or a composite structure).

Using these four elements of OOP we have provided the engineer, who works with real-world physical objects, an environment in which he can declaratively model the objects in his or her problem space and the objects will retain their identify in the solution space. In other words, although various portions of the system model the characteristics of analysis differently (e.g., the entities of the *h-p* finite element method are very small finite elements of the objects in the model), the results of the analysis are posted in the same form as the engineer's description.

For example, the kernel hierarchies in the Engineering Design Environment might consist of one of the two superclasses MATERIAL and STRUCTURE, STEEL defined as a subclass of MATERIAL, BEAM and COLUMN defined as subclasses of STRUCTURE, STEEL-BEAM defined as a subclass of both STEEL and BEAM, and STEEL-COLUMN defined as

a subclass of both STEEL and COLUMN. Using the concept of element structure (part of) the engineer could assemble an instance of the class BRACKET comprised of two instances of the class STEEL-COLUMN and one instance of the class STEEL-BEAM.

In the context of an Automated Design Environment the Object Oriented Programming provides a very natural and well defined way of representing structural objects, finite element entities, and even humans (experts) involved in the design process.

5.3 Expert Systems and Knowledge Engineering

In contrast to algorithmic procedures, computer tools for handling shallow knowledge took considerably longer to develop. However, three decades of research in the area of knowledge engineering or "artificial intelligence" have resulted in the development of effective methods and tools for handling shallow knowledge and simulating human reasoning and the learning processes. Presently the best known and widely implemented tools for simulating the human expertise are expert systems.

Expert systems are software packages utilizing expert knowledge coded into rules to find solutions to problems which cannot be solved algorithmically, or in which the algorithmic procedure can lead to a blind search. Expert systems are usually composed of three sections: a "fact base" which contains active knowledge presented in the form of rules, the "inference base" which also contains active knowledge presented in the form of rules, and the "inference engine" which applies rules to generate new facts and arrive at required conclusions. The most popular modes of operation of the inference engine are "forward chaining", "backward chaining", and their combinations [17,37].

The first generation of expert systems could be characterized as a rule–based advisory systems for various applications. A good example is one of the first expert systems to gain acknowledgement is MYCIN. MYCIN has a backward chaining inference engine and a knowledge base consisting of rules for determining the best combination of antibiotic medication to be prescribed for a patient with a viral infection. In the field of engineering applications the corresponding expert system is SACON [12]. These first expert systems usually consisted of a few hundred IF THEN rules with fact bases stored in the form of simple statements, e.g., "The speed of the car is 100 mph." This simple form of the knowledge base limited the flexibility of expert systems and made the development of larger systems (with more than a few hundred rules) practically impossible. An additional disadvantage of these systems was the fact that they were usually developed in one of the dialects of the LISP computer language, requiring specialized (and expensive) computers which were in general incompatible with languages traditionally used in algorithmic operations. Due to the above reasons, the expert systems of this first generation never gained real popularity.

Recent years, however, have brought several major advances to the field of artificial intelligence, which allow practical development of powerful expert systems with the capability of interacting with other programs. These advances include:

- The introduction of general purpose "shells" (expert system development tools) that can be used to create "user defined" expert systems for arbitrary applications.
- The application of more conventional languages (C, PROLOG, etc.) rather than LISP in the development of expert system software.
- The use of knowledge bases with a more convenient structure than rules, e.g., semantic networks, frames, or variations of object–oriented programming.

The general purpose shells are the pieces of software designed to serve as the expert system building tools for generic applications: the expert system developer uses shell commands to build the knowledge base, while the actual user utilizes the expert system to make decisions. Presently there exists a variety of expert system shells available on the market, with different capabilities and prices ranging from a few hundred dollars to about $50,000. Some of the most popular systems will be reviewed further in this section.

As mentioned before, the first expert systems were developed in the LISP language, which indeed is a very powerful language for knowledge engineering, but unfortunately requires rather expensive LISP machines and is difficult to couple with other languages. These inconveniences were the reason that most of the currently marketed expert system shells are developed in one or more flexible languages, like C or PROLOG. Especially the C language (or its object–oriented versions, C++ or CLIPS) is very popular, because it is highly portable and, more importantly, it is a mechanism for direct interaction of expert systems with programs developed in other languages such as PASCAL or FORTRAN.

Another significant advance in knowledge engineering is the improved structure of knowledge bases. Such techniques as semantic networks [17,31], blackboarding [37,75], or object–oriented capabilities enable relatively easy construction of large and versatile expert systems.

As mentioned before, there exists a variety of expert system shells available on the market. A brief summary of some of the most popular software is compiled below. The information provided here is based on various sources and is only for the purpose of providing a general orientation.

ART-IM — is the expert system shell developed by the Inference Corp. It was developed in the C language, runs on workstations and powerful PCs, and can be integrated with programs written in other conventional languages (FORTRAN, PASCAL, etc.). The inference engine is based

on forward chaining. The shell has object–oriented capabilities and allows for application of certainty factors in rules (as opposed to simple TRUE or FALSE statements).

COPERNICUS — is the set of software tools for developing expert systems, developed by Teknowledge. It is written in the C language and can be integrated with other applications. The inference engine supports forward chaining, backward chaining, nonmonotonic reasoning, etc. Certainty factors for rules are allowed. The system also supports object–oriented programming.

EXSYS — is the expert system developed by Exsys, Inc. It is a relatively simple rule–based system, developed in the C language. It runs on workstations and PCs and can be integrated with programs written in other conventional languages (FORTRAN, PASCAL, etc.). The inference engine has both forward and backward chaining capabilities. Certainty factors for rules can also be used.

GOLDWORKS II — is the expert system shell developed by Goldhill Computers, Inc. It runs on workstations and powerful PCs and can be integrated with programs written in C or equipped with an Application Programming Interface (API). The inference engine has forward and backward chaining capabilities, goal–directed chaining and other options. The knowledge base supports rules, frames, and object–oriented capabilities. Certainty factors for rules can be used.

KAPPA — is the expert system shell developed by Megaknowledge, Inc. (currently owned by Intellicorp). It was written in C, runs on powerful PCs, and can be integrated with programs written in other conventional languages (FORTRAN, PASCAL, etc.). The inference engine has both forward and backward chaining capabilities, as well as more advanced options, like a goal–driven approach. The knowledge base has object–oriented capabilities.

MERCURY — is the knowledge base environment developed by Artificial Intelligence Technologies, Inc. The inference engine has both forward and backward chaining capabilities and their combinations. The knowledge base includes rule–based and object–oriented programming. The expert systems developed can be integrated with other software, in particular with data bases, etc.

NEXPERT OBJECT is the expert system shell developed by Neuron Data Inc. It was written in C language, runs on the majority of workstations and powerful PCs, and can be integrated with programs written in other conventional languages (FORTRAN, PASCAL, etc.). The inference engine has both forward and backward chaining capabilities, and the knowledge base has object-oriented capabilities. Interestingly, NEXPERT shell can be coupled with NESTOR neural network with pattern recognition capabilities.

1st-CLASS FUSION is an expert system software developed by 1st-Class Expert Systems, Inc. It runs on PCs and DEC VAX. It can be integrated with other softward (dBASE, LOTUS) or programs written in C or PASCAL. The inference engine has both forward and backward chaining capabilities. Knowledge representation is rule-based but also offers an interesting example-based option.

The above list represents only a fraction of the total number of expert system software currently available for generic applications. Even this brief review clearly shows that the expert system technology has established its position as the primary tool for knowledge engineering.

5.4 Knowledge Acquisition

One of the primary and most difficult tasks in building expert systems is constructing the actual knowledge bases. In most of today's expert systems this amounts to formulating rules of the IF – THEN – ELSE format. These rules are introduced into the system using the language specific to a given expert system tool. In practice this requires direct cooperation of a knowledge engineer with the actual expert or, alternatively, training this expert in effective interaction with the expert system software. With this approach, the capabilities of an expert system never exceed the common knowledge of human experts involved in the creation of a knowledge base.

There exist, however, new developments in the area of knowledge acquisition, which promise more flexible behavior of expert systems. In the simplest case, it is possible to construct rules from examples provided by the human expert. Such an option is available, for example, in the 1st-CLASS Expert system shell. In a more general scenario, it is possible to create new rules from previously solved examples. For example, in the process of structural optimization of a certain class of aerospace structures, previously solved problems can be used to improve the optimization procedure for new examples.

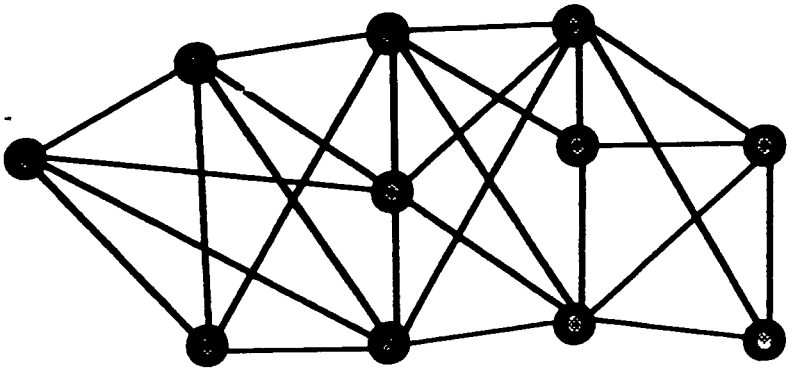


Figure 5.1: Neurons and connections in neural network.

Automation of this learning capability will require intelligent classification of examples and extraction of essential facts relevant to the problem under consideration. It can be expected that in the near future such capabilities will be available as extensions to the existing knowledge engineering software. Then the performance of expert systems will actually be improving during their application and may eventually exceed the capabilities of human experts.

5.5 Neural Networks

Neural networks represent a very special application of knowledge engineering. They consist of simulated neurons connected together in a way simulating, in a very simplified way, the connections in human brains. Therefore, the kinds of problems that neural networks can solve are also those that people can do well, in particular:

- association,
- pattern recognition,
- qualitative evaluation, and
- learning.

The concept of a neural network is presented graphically in Fig. 5.1. Each neuron has several input connections and output conncetions. The input connections are both excitatory

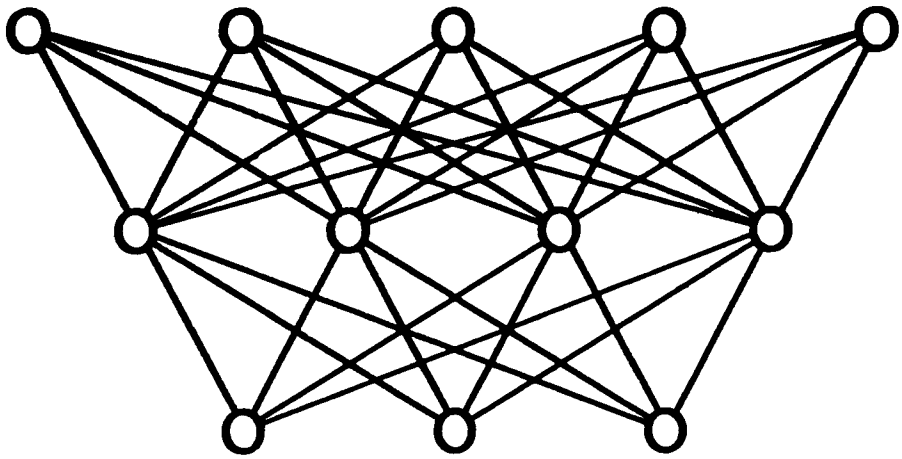


Figure 5.2: A three-layered neural network.

and inhibitory. The response of the neuron to the stimulations is defined by the activation function, which balances stimulations from all inputs (several different formulas are available) to produce the output. In more complex cases the activation function also uses the previous state of the neuron.

In practical applications the neurons and connections are constructed in a certain organized fashion. For example, in the popular type of feed-forward network there exist three layers of neurons: input layer, hidden layer and output layer (Fig. 5.2). Moreover, the actual implementations may be hardware-based (with hardware elements representing each neuron and connection) or implemented as a computer simulation.

Because of their structure, the neural networks can produce certain reactions (output) to different combinations of input stimuli—for example, select from a predefined set a pattern closest to an incomplete input pattern. A more detailed discussion of these capabilities can be found in references [1,5,20,21,42].

An important feature of most neural nets is that their responses are not predefined. Instead, a learning mechanism is involved. A network has to be trained to properly adjust weighing factors in activation functions. There are two basic kinds of training, namely:

- supervised training, and
- unsupervised training.

In supervised training the weighing functions are adjusted by comparison of the network output with ideal results for a certain predefined set of examples. This is usually done

by the application of the so-called Delta learning rule [42]. In unsupervised training the connections are adjusted by solving a variety of examples with no predefined results. The learning algorithm in this case is usually based on variations of Hebb's rule, which actually represents a theoretical model of biological associative memory [42].

In the context of automated computational mechanics, the neural networks can be used to:

1. Identify patterns in solution processes, design applications, optimization processes, etc. This information can be used to improve the efficiency of solving new examples by utilizing patterns detected on examples previously solved.

2. Reconstruct proper parameters in incomplete data bases, input decks, etc.

It seems that at the present time the possibilities of practical application of neural networks in automated computational mechanics are rather limited. This is due to the still rather experimental nature of neural networks, limited complexity of problems (patterns) that can be handled by today's neural nets as well as the difficulties in the direct interaction of algorithmic programs with neural network technology.

It should be noted, however, that neural network technology is currently being coupled with expert systems software—for example, the recently announced coupling of NEXPERT OBJECT expert system shell with NESTOR neural network. Therefore, once the finite element methodology is effectively interfaced with expert system software, it will automatically benefit from further progress in knowledge engineering, including neural network technology.

6 Methods, Concepts, and Algorithms for Automated Computational Mechanics

In this section we will discuss methods, concepts, and algorithms related to automation of the design processes outlined in Section 3. In particular, a critical look at existing methods and some new concepts of automated computational mechanics will be presented. Several of these developments were used in the implementation of an automated, coupled finite element expert system environment (Sections 7 and 8).

6.1 Selection of a Mathematical Model

The selection of the mathematical model of the structure requires considerable theoretical background and experience. The choice of the actual model depends on several factors,

some of them highly quantifiable, others somewhat heuristic. The principal groups of factors affecting the selection of a mathematical model are:

1. **The actual data of the problem, namely the shape of the structure, the types of materials, loads, supports, interactions with the environment and other structures, time scales, etc.**

2. **The specific aspects of behavior that the designer feels will influence the final product.**

3. **The actual theoretical, technical, and computational capabilities available to the designer.**

Until very recently, the above decisions were primarily heuristic. For example, a typical decision whether a structural member should be represented as a shell or modeled as a solid body was based on a simple estimate of the proportion of its dimensions. Only after complete analysis could the experienced engineer examine results and possibly modify the model (for example, concentration of stress in corners of a shell indicates a need for a full three–dimensional model). There exists, however, a considerable theoretical potential toward more precise *a priori* identification of a proper mathematical model.

In this section we discuss two possible systematic approaches to this problem, namely:

- asymptotic theories, dealing with the selection of a general model (solid, plate, shell, etc.), and
- hierarchical methods for systematic construction of mathematical models of increasing accuracy.

6.1.1 Asymptotic Analysis of a Family of Elasticity Problems

The methods of asymptotic analysis are an attempt to provide a rigorous approach to constructing mathematical models of solid structures. This theory has been applied to certain linear problems in elasticity and structural mechanics [23,24] and, in general, should be applicable to a broader class of problems in solid mechanics. In this section we will discuss an application of asymptotic analysis to determine whether a solid body, presented in Fig. 6.3, can be modeled by beam theory or should be modeled as a three–dimensional solid.

Let ϵ denote an arbitrary positive parameter (for asymptotic expansion) and consider a three–dimensional body Ω^ϵ made up of a homogeneous isotropic linear elastic material. (The role of the expansion parameter will become apparent subsequently.) The body under consideration is assumed, for purposes of the discussion, to be a beam–like structure with two

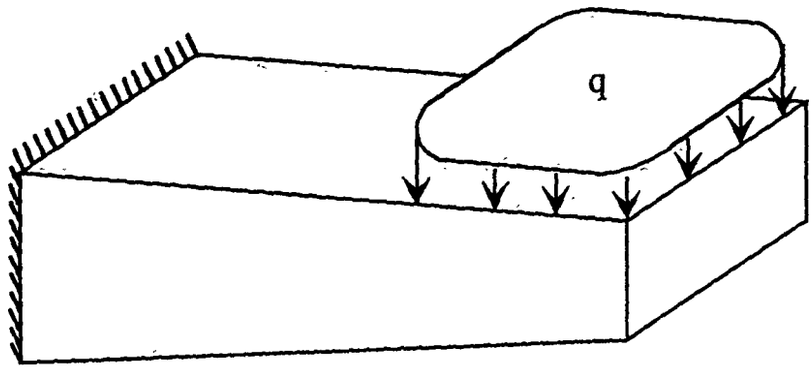


Figure 6.3: A three-dimensional solid body.

dimensions much smaller than the third. The material from which the body Ω^ϵ is constructed has a modulus of elasticity E^ϵ and Poisson's ratio ν^ϵ. We denote by $f_i^\epsilon(x^\epsilon)$ (resp. $g_i^\epsilon(x^\epsilon)$) the i-th component of the volume (resp. surface) density of applied body forces (resp. surface tractions) at point $x^\epsilon \in \Omega^\epsilon$ (resp. $x^\epsilon \in \Gamma_1^\epsilon$), and we introduce the following function spaces:

$$V^\epsilon = \left\{ v^\epsilon = (v_i^\epsilon) \in \left[H^1(\Omega^\epsilon)\right]^3 : \int_{\Omega^\epsilon} x^\epsilon \times v^\epsilon = 0 \text{ in } \Gamma_0^\epsilon \right\},$$

$$\Sigma^\epsilon = \left[L^2(\Omega^\epsilon)\right]_s^9 = \left\{ \zeta = (\zeta_{ij}) \in \left[L^2(\Omega^\epsilon)\right]^9 : \zeta_{ji} = \zeta_{ij} \right\}$$

Here V^ϵ is the space of motions of finite energy with no allowable rigid rotations and Σ^ϵ is the space of stress (strain) tensors. We equip these spaces with the usual norms. Mathematically, the constitutive law is characterized by the symmetric automorphism $A^\epsilon : \mathbb{R}_s^9 \to \mathbb{R}_s^9$, defined by:

$$Y_{ij} = (A^\epsilon x)_{ij} = \frac{1+\nu^\epsilon}{E^\epsilon} X_{ij} - \frac{\nu^\epsilon}{E^\epsilon} X_{pp}\delta_{ij},$$

where

$$\mathbb{R}_s^9 = \left\{ \zeta = (\zeta_{ij}) \in \mathbb{R}^9 : \zeta_{ij} = \zeta_{ji} \right\}$$

Then the Hellinger-Reissner formulation of the three-dimensional elaticity problem for the weakly clamped beam (other boundary conditions shall be considered later) is the following:

$$\left\{ \begin{array}{l} \text{Find } (\sigma^\epsilon, u^\epsilon) \in \Sigma^\epsilon \times V^\epsilon, \text{ (a stress-displacement pair) such that} \\ \displaystyle\int_\Omega \epsilon (A^\epsilon \sigma^\epsilon)_{ij} \zeta_{ij}^\epsilon - \int_\Omega \epsilon \gamma_{ij}^\epsilon(u^\epsilon) \zeta_{ij}^\epsilon = 0, \ \forall\, \zeta^\epsilon \in \Sigma^\epsilon, \\ \displaystyle\int_\Omega \epsilon \sigma_{ij}^\epsilon \gamma_{ij}^\epsilon(v^\epsilon) = \int_\Omega \epsilon f_i^\epsilon + \int_{\Gamma_1^\epsilon} g_i^\epsilon, \ \forall\, v^\epsilon \in V^\epsilon, \end{array} \right. \tag{6.1}$$

where γ_{ij}^ϵ is the strain tensor:

$$\gamma_{ij}^\epsilon(v^\epsilon) = \frac{1}{2}\left(\partial_i^\epsilon v_j^\epsilon + \partial_j^\epsilon v_i^\epsilon\right), \ \forall\, v^\epsilon \in V^\epsilon$$

Here and below we reproduce the formulations of Trabucho and Viaño [78]. As a consequence of a result of Brezzi [18], for mixed formulations, and by using Korn's inequality the existence of a unique solution of problem (6.1) is guaranteed.

The dependence of $(\sigma^\epsilon, u^\epsilon)$ on the parameter ϵ, related to the size of the cross section, is rather complex. In order to study its behavior when ϵ becomes small, we shall use a change of variable technique in such a way that ϵ shows up in the governing equations in an explicit form. The specific change of variable considered here is the one proposed in Bermudez-Viaño [13] and is analogous to the one of Ciarlet-Destuynder [23].

We shall now consider the following function spaces:

$$V = \left\{ v = (v_i) \in \left[H^1(\Omega)\right]^3 : \int_\Omega v = \int_\Omega x \times v = 0 \text{ on } \Gamma_0 \right\}$$

$$\Sigma = \left[L^2(\Omega)\right]^9_s$$

equipped with the usual norms.

Moreover, we shall assume that the elasticity coefficients do not vary with ϵ, that is:

$$E^\epsilon = E \; ; \; \nu^\epsilon = \nu \; ; \; \forall \, \epsilon > 0.$$

We shall also denote by A^0 the following symmetric automorphism defined on the space $\mathbb{R}^4_s = \{(\zeta - (\zeta_{\alpha\beta}) \in \mathbb{R}^5 : \zeta_{\alpha\beta} = \zeta_{\beta\alpha}\}$

$$Y_{\alpha\beta} = \left(A^0 x\right)_{\alpha\beta} = \frac{1+v}{E} X_{\alpha\beta} - \frac{v}{E} X_{\mu\mu}\delta_{\alpha\beta}$$

For any $(\sigma, \zeta) \in \Sigma \times \Sigma$ and any $v \in V$ we define the following virtual work forms for the Hellinger-Reissner principle:

$$a_0(\sigma, \zeta) = \frac{1}{E} \int_\Omega \sigma_{33}\eta_{33}$$

$$a_2(\sigma, \zeta) = \int_\Omega \left[\frac{\nu(1+v)}{E} \sigma_{3\beta}\zeta_{3\beta} - \frac{v}{E} \left(\sigma_{33}\zeta_{\mu\mu} + \sigma_{\mu\mu}\zeta_{33}\right) \right]$$

$$a_4(\sigma, \zeta) = \int_\Omega \left(A^0\sigma\right)_{\alpha\beta} \zeta_{\alpha\beta}$$

$$b(\zeta, v) = -\int_\Omega \zeta_{ij}\gamma_{ij}(v)$$

$$f_0(v) = -\int_\Omega f_i^0 v_i - \int_{\Gamma_1} g_i^0 v_i.$$

With this notation in mind, the following result may be shown:

Theorem [Trabucho and Viaño [78], p. 307]: Let $(\sigma(\epsilon),\ u(\epsilon)) \in \Sigma \times V$ be the stress-displacement pair obtained from the solution $(\sigma^\epsilon,\ u^\epsilon) \in \Sigma^\epsilon \times V^\epsilon$ of (6.1), then $(\sigma(\epsilon),\ u(\epsilon))$ is the unique solution of the following problem:

$$\begin{cases} \text{Find } (\sigma(\epsilon)\ ,\ u(\epsilon)) \in \Sigma \times V, \text{ such that:} \\ a_0(\sigma(\epsilon), \zeta) + \epsilon^2 a_2(\sigma(\epsilon), \zeta) + \epsilon^4 a_4(\sigma(\epsilon), \zeta) + b(\zeta, u(\epsilon)) = 0, \ \forall\, \zeta \in \Sigma \\ b(\sigma(\epsilon), v) = F_0(v), \ \forall\, v \in V. \end{cases}$$

Following a standard technique for variational problems with small parameters (see Lions [44]), we shall suppose that we may write, at least formally,

$$(\sigma(\epsilon), u(\epsilon)) = (\sigma^0, u^0) + \epsilon^2(\sigma^2, u^2) + \epsilon^4(\sigma^4, u^4) + (\eta(\epsilon), \omega(\epsilon))$$

where $(\eta(\epsilon), \omega(\epsilon)) \to 0$ as $\epsilon \to 0$, in an appropriate space. Identifying the coefficients with the same powers in ϵ, we may characterize the terms $(\sigma^{2p} u^{2p})$, $p = 0, 1, 2$ as the solution of the following system of equations valid for all $\zeta \in \Sigma$ and all $v \in V$:

$$\begin{cases} a^0(\sigma^0, \zeta) + b(\zeta, u^0) = 0 \\ b(\sigma^0, v) = F_0(v) \end{cases} \tag{6.2}$$

$$\begin{cases} a_0(\sigma^2, \zeta) + b(\zeta, u^2) = -a_2(\sigma^0, \zeta) \\ b(\sigma^2, v) = 0 \end{cases}$$

$$\begin{cases} a_0(\sigma^4, \zeta) + b(\zeta, u^4) = -a_2(\sigma^2, \zeta) - a_4(\sigma^0, \zeta) \\ b(\sigma^4, v) = 0 \end{cases} \tag{6.3}$$

A main result of this type of exercise, not immediately apparent from (6.2) and (6.3), is that component of the expansion that becomes relevant (mathematically well posed with a nontrivial solution) is data dependent. That is, *by identifying the data in the problem the theory appropriate for the problem at hand falls out automatically from the asymptotic formula.* In other words, if the boundary conditions and loading data and geometry of Ω suggest that this structure can be adequately treated as a thin elastic plate or a thin elastic shell, then it is be possible to automatically determine this from a simple analysis of the data itself.

6.1.2 Hierarchical Models in Structural Mechanics

For a typical problem in structural mechanics, there usually exist a variety of mathematical models representing the mechanical behavior of the structure. As a general rule, more comprehensive models are usually mathematically more complicated and computationally more expensive. Therefore one of the essential tasks in computational mechanics is the selection of an appropriate model by balancing two criteria: *reliability* and *effectiveness.*

These criteria are defined by Bathe, et al. [11] by the introduction of a *very comprehensive* mathematical model, i.e., a model that fully (or rather to the best of our knowledge) represents the system under consideration. The mathematical model selected is reliable if

the response predicted is within a selected level of accuracy, measured with respect to the response of the very comprehensive mathematical model. The model is effective if it yields the required response at minimum cost.

Unfortunately, there exist no single consecutive hierarchy of models in solid mechanics. Even if one restricts the considerations to purely mechanical behavior (no thermal, electromagnetic, chemical effects, etc.), there exist several distinct aspects or classes in which some kind of sequential hierarchy can be established.

The first hierarchy is related to the geometric model of the structure. Here the hierarchy is defined by the number of constraints and assumptions introduced with respect to fully three-dimensional behavior. In general this hierarchy includes:

- very thin one-dimensional models (cables)
- one-dimensional beam-type models with such subclasses as
  - Bernoulli beams
  - Timoshenko beams
  - generic curved beams, etc.
- very thin two-dimensional models (membranes)
- two-dimensional shell or plate models, with such subclasses as
  - Kirchhoff plates (shells)
  - Reissner-Mindlin plates, etc.
- fully three-dimensional models

In general, the simplified models in this group reduce the number of dimensions and unknowns in the problem by introduction of additional assumptions (constraints) regarding the behavior of the model, usually at the cost of increating the mathematical complexity of the equations. For example, in Kirchhoff plate theory, it is assumed that the fiber normal to the plate surface remains normal and inextensible during deformation. Then it suffices to consider only a two-dimensional model with one unknown, namely normal deflection of the plate (or three displacements for shells). The Reissner-Mindlin plate model is a higher-order model than Kirchhoff, but at the cost of introducing an additional variable, namely an in-plane twist. Further upgrades of these models can be obtained by introducing rotations of the normal fiber and its extensions as independent variables. Note that the above classification is further complicated by the introduction of intermediate classes such as shallow shells, for example.

The second essential aspect defining the hierarchy of mathematical models is the kinematic definition of the deformation of the body. There are three basic categories in this class, namely:

- infinitesimal deformation theory, defined by linear differential equations,
- large rotation, small strain theory, and
- fully nonlinear large rotation theory.

For selected structural types such as shells, this division may include additional intermediate classes such as large deflection/small rotation theory, etc.

A chart representing a general hierarchy of constitutive models is presented in Fig. 6.4. Although this general hierarchy is relatively simple, establishing a hierarchy of mathematical formulations is much more difficult, because there usually exist several different mathematical representations for the same constitutive behavior. A good example is the theory of plasticity, where there exists a variety of formulations based on different principles, different selection of yield criteria, etc.

The value of a hierarchical approach in the selection of the mathematical model of the structure is that it provides a systematic way of selecting the model by examining the essential features of the physical problem. Moreover, this approach is very useful in the verification and refinement of the mathematical model *after* solving the problem by a numerical method. This approach is discussed in more detail in Section 6.4.

6.2 Adaptive Computational Techniques

By adaptive computational techniques we mean a class of computational methods that adapt their parameters to changing characteristics of the solution during the solution during the solution process. These techniques include:

- adaptive mesh techniques,
- adaptive time stepping in time–dependent problems,
- adaptive selection of load steps in nonlinear problems, and
- adaptive selection of implicit and explicit zones and other zonal methods.

Most of these methods are based on solid mathematical foundations and are implemented in an algorithmic fashion. These techniques have received a great deal of attention from researchers in recent years. However, in virtually all these techniques certain heuristic knowledge is necessary for maximum performance and reliability in practical applications.

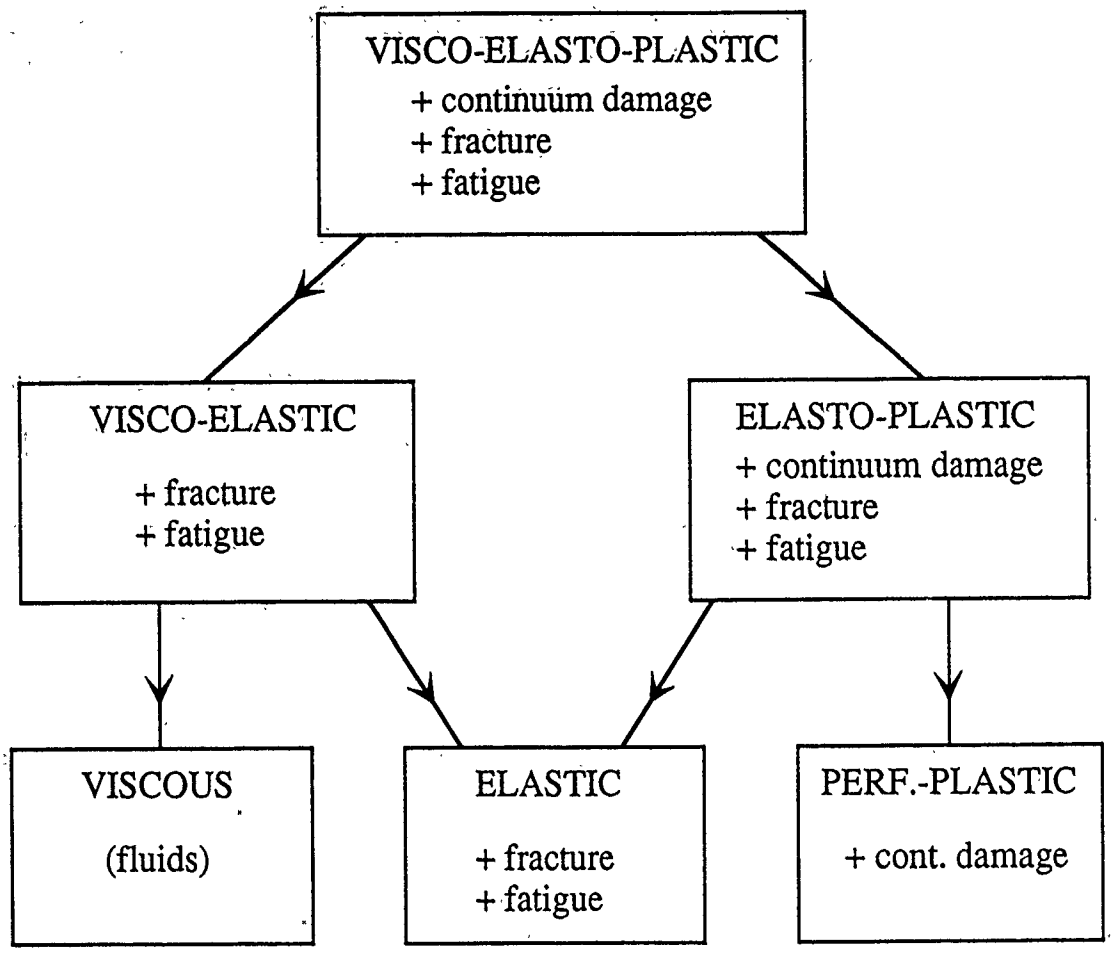


Figure 6.4: Classification of constitutive models in structural mechanics.

In this section we present the theoretical background and methodology of several adaptive computational methods developed and implemented at The Computational Mechanics Company, Inc. Some practical applications and numerical results will be discussed in Section 8.

6.2.1 Error Estimation and Adaptive Mesh Refinements

Adaptive mesh refinement, based on rigorous error estimates, enables resolution of a basic question of modern computational mechanics: *what is the accuracy of the numerical solution?* Equivalently, *how good are the answers?* Answers to these questions are crucial in the design of mathematically sound adaptive schemes: to systematically reduce error by adaptively changing mesh sizes or spectral orders of approximations, one must obviously have some means to judge the distribution of error in a numerical solution.

In this section, we will outline some of the most important results related to error estimation and adaptive techniques for *h-p* finite element methods. Details of their derivations and proofs can be found in references [29,30]. The theoretical background of *h-p* finite element methodologies and associated data structures is summarized in Appendix A.

There are, in general, several classes of error estimators for the *h-p* finite element method, including:

1. *Element Residual Methods.* Here the residual in a numerical solution (the function defining the measure of how much the approximate solution fails to satisfy the governing differential equations and boundary conditions) is computed over each element and used as data in special local (elementwise) BVPs for the local error e_h.

2. *Duality Methods.* These methods, valid for self–adjoint elliptic problems, use the duality theory of convex optimization to derive upper and lower bounds of the element errors.

3. *Subdomain Residual Methods.* Here the local BVB for the error in a given element is formulated over a patch of elements surrounding this element.

4. *Interpolation Methods.* These methods use the interpolation theory of finite elements in Sobolev norms to produce rapid (and sometimes crude) estimates of the local error over individual elements.

5. *Post–Processing Methods.* Here an estimate of the error is obtained by comparing a post–processed version of the approximate solution with the approximate solution.

We will present basic results related to these error estimators for the following model problem. Consider a bounded domain $\Omega \subset \mathbb{R}^2$ with boundary $\Gamma = \partial\Omega$ consisting of two disjoint portions Γ_u and Γ_t on which a function $u = u(x_1, x_2) = u(\boldsymbol{x})$ is sought; i.e., we wish to

$$\left.\begin{array}{ll} \text{Find } u \in \overline{u} + V & \text{such that} \\ B(u,v) = L(v) & \forall\, v \in V \end{array}\right\} \tag{6.4}$$

where V is the space of admissible functions,

$$V = \left\{ v \in H^1(\Omega) | v = 0 \text{ a.e. on } \Gamma_u \right\}$$

and $B(\cdot,\cdot)$ and $L(\cdot)$ are the bilinear and linear forms,

$$\begin{aligned} &B : V \times V \to \mathbb{R} \\ &B(u,v) = \int_\Omega (a \nabla u \cdot \nabla v + buv) dx \\ &L : V \to \mathbb{R} \\ &L(v) = \int_\Omega fv dx + \int_{\Gamma_t} gv ds \end{aligned}$$

Here $\overline{u}_0$ is any $H^1(\Omega)$ function with the property that its trace $\gamma\overline{u}_0$ on Γ_u coincides with a prescribed function u_0; $v = 0$ in the definition of V is understood in the sense of its trace on $\partial\Omega$, and the coefficients $a = a(\boldsymbol{x})$ and $b = b(\boldsymbol{x})$ satisfy, for any $\boldsymbol{x} \in \Omega$,

$$0 < \underline{a} \leq a(\boldsymbol{x}) \leq \overline{a},\ 0 < \underline{b} \leq b(\boldsymbol{x}) \leq \overline{b}$$

where $\underline{a}$, $\overline{a}$, $\underline{b}$, and $\overline{b}$ are constants. Throughout the following developments, C shall denote a generic positive constant independent of the finite element mesh, unless specifically noted otherwise. In exceptional cases, special notations for constants shall be used. Additionally, problem (6.4) is equivalent to the following minimization problem,

$$\left.\begin{array}{ll} \text{Find } u \in \overline{u}_0 + V & \text{such that} \\ J(u) \leq J(w) & \forall\, w \in \overline{u}_0 + V \end{array}\right\}$$

where $J: H^1(\Omega) \to \mathbb{R}$ is the *total potential energy functional* defined by

$$J(v) = \frac{1}{2} B(v,v) - L(v) \tag{6.5}$$

A Priori Interpolation Error Estimate

An interpolation error estimate is a relatively cheap and rather crude estimate, which can effectively be used as an indicator for adaptive mesh refinement. For an arbitrary h-p mesh it has the form:

$$\left\| u - u_p^h \right\|_{1,K} \leq C h^{\mu-1} p^{r-1} \|u\|_{r,K} \tag{6.6}$$

where $\mu = \min(p+1, r)$ and u_p^h is the finite element solution to problem (6.4).

Residual Error Estimate

The idea of a residual error estimator is based on the assumption that the actual error of the solution u_H:

$$e_H = u - u_H$$

is sufficiently well estimated by the error between u_H and the solution on enriched mesh u_h

$$E_H = u_h - u_H$$

The energy norm of this error can be estimated [30] by

$$|||E_H||| \leq C \left\{ \sum_K |||\varphi_K|||_K^2 \right\}^{\frac{1}{2}}$$

Here C is a constant dependent on the order of approximation. It can be assumed to be equal to 1.0 for $p = 1$ and 1.25 for $p > 1$. φ_K is a local element error contribution which is a solution of a local problem:

$$\left.\begin{aligned} &\text{Find } \varphi_K \in M_h(K) \text{ such that} \\ B_K(\varphi_K, v_h) &= \int_K r_h v_h dx + \int_{\partial K \setminus \partial\Omega} \frac{1}{2} \left[\left[a \frac{\partial u_H}{\partial n_K} \right]\right] v_h ds \\ &+ \int_{\partial K \cap \Gamma_\varepsilon} \left(g - a \frac{\partial u_H}{\partial n_K} \right) v_h ds \\ &\forall v_h \in M_h(K) \end{aligned}\right\}$$

where r_h is a local element residual, $[v]$ denotes the jump in v across the interelement boundary, n is the outward normal to the element boundary, and $M_h(K)$ is the space of bubble functions defined over element K. For details on the derivation of this error estimator, see reference [30]. It is of importance to note that, although the fine mesh solution is formally used in the derivation of the residual estimator, the problem is never solved on the final mesh. The only time consuming operation is the calculation and accumulation of local, elementwise contributions to the error indicator.

Error Estimator Base on Duality Theory

Consider the boundary value problem (6.4) with energy defined by equation (6.5). Introducing the space

$$Q = \left\{ s \in L^2(\Omega) | \text{ div } s \in L^2(\Omega) \right\}$$

and the set

$$K^* = \{ s \in Q | s \cdot n = g \text{ on } \Gamma_t \}$$

we can define the following dual problem

$$\left.\begin{array}{l} \text{Find } r \in K^* \text{ such that} \\ J^*(r) = \sup_{s \in K^*} J^*(s) \end{array}\right\}$$

where $J^*(s)$ the *complementary energy function*

$$J^*(s) = \frac{1}{2} \int_\Omega \left[a^{-1} s \cdot s + b^{-1} (f + \text{div } s)^2 \right] dx \tag{6.7}$$

An *a posteriori* error estimator is based on the fact that the solutions of both the primal and the dual problem satisfy the relationship:

$$J(u) = \inf_{v \in V} J(v) = \sup_{s \in Q} J^*(s) = J^*(r)$$

Then it can be shown that:

$$|||u - u_h|||^2 \leq 2J(u_h) - 2J^*(r_h)$$

and

$$|||r - r_h|||^2 \leq 2J(u_h) - 2J^*(r_h)$$

Thus, the errors in both the primal and the dual approximation are each bounded in their respective global energy norms by the difference of the approximate primal and complementary energies.

A direct calculation and Green's formula reveal that

$$2J(u_h) - 2J^*(r_h) = \int_\Omega \left(a^{-1} (r_h - a \nabla u_h)^2 + b^{-1} (f + \text{div } r_h - b u_h)^2 \right) dx$$

Suppose that Ω is given a partition $\mathcal{Q}_h$ of finite elements over which primal and dual approximations are calculated. Then, for element $K \in \mathcal{Q}_h$, we take as the *local error indicator*

$$\theta_K^2 = \int_K \left(a^{-1} (r_h - a \nabla u_h)^2 + b^{-1} (f + \text{div } r_h - b u_h)^2 \right) dx$$

In general, for linear elliptic boundary-value problems, the quantities θ_K give remarkably good estimates of the local error in the energy or complementary energy over each finite element. Globally, we always have

$$\left.\begin{aligned} |||u - u_h|||^2 \\ |||r - r_h|||_*^2 \end{aligned}\right\} \leq \sum_{K \in \Omega_h} \theta_K^2$$

Subdomain–Residual Methods

Subdomain residual methods employ the general ideas of residual methods to calculate local error indicators by covering the domain Ω with subdomains Ω_i, and constructed by the selection of a patch of elements around element i. For each patch, we introduce a local projection problem:

$$\left.\begin{aligned} &\text{Find } \psi_i \in H_0^1(\Omega_i) \qquad \text{such that} \\ &B(\psi_i, v) = B(e_h, v) \quad \forall\, v \in H_0^1(\Omega_i) \end{aligned}\right\} \tag{6.8}$$

Moreover, we have the estimate that:

$$|||e_h||| \leq C \left(\sum_{i=1}^{N} |||\varphi_i|||^2 \right)^{\frac{1}{2}}$$

Notice that the right hand side in the local projection problem (6.8) is a function of the finite element residual:

$$B(e_h, v) = \int_{\Omega_i} f v dx - B(u_h, v)$$

Representing the contribution of the i-th subdomain Ω_i to the global estimate in the form of a sum over the elements constituting the patch

$$|||\varphi_i|||^2 = \sum_{K \subset \Omega_i} |||\varphi_i|||_K^2$$

we introduce for each element K a set of up to four indices identifying its linear degrees-of-freedom and, consequently, the patches Ω_i containing the element:

$$\sigma(K) = \{i | K \subset \Omega_i\}$$

We can then rewrite the estimate in the form

$$|||e_h||| \leq C \left(\sum_K \theta_K^2 \right)^{\frac{1}{2}}$$

where the element error indicators θ_K are defined as

$$\theta_K^2 = \sum_{i \in \sigma(K)} |||\varphi_i|||_K^2$$

Thus, in order to calculate the error indicator for an element K, one has to identify the subdomains containing the element, solve the local subdomain problem (6.8) for each of the subdomains, calculate the element contribution to each of the $|||\varphi_i|||^2$, and sum them up over the subdomains.

The local projection problem in practice is solved using a higher order approximation.

A Posteriori Error Estimator Based on the Interpolation Error Estimates

The interpolation error estimate may be used as an *a posteriori* error estimate, provided the second-order derivatives needed to evaluate the seminorm on the right hand side of (6.6) are calculated using the finite element solution and some kind of postprocessing. This type of error approximation has two attractive features:

(a) The interpolation is a local quantity defined for individual elements.

(b) The interpolation error is problem-independent; it depends only on u and the interpolation properties of finite element meshes.

The constant on the right hand side of (6.6) is often set to unity and a local error indicator is assumed in the form

$$\varphi_K = \frac{h_K}{p_K} |u|_{2,K}$$

These error indicators provide an estimate of the local behavior of the interpolation error. The global quantity

$$\left(\sum_K \varphi_K^2 \right)^{\frac{1}{2}}$$

is rather meaningless, unless precise estimates of the neglected constants are available.

A Posteriori Error Estimator Based on Postprocessing

Perhaps the most straightforward approach toward estimating the error *a posteriori* is to replace the exact solution u and its derivatives with some postprocessed values calculated using *data* to the problem as well as the finite element solution u_h. The postprocessed solution should then exhibit superior accuracy and convergence properties over the finite element solution itself.

For instance, an error estimator (indicator) for an element K, based on the energy norm, may be of the form

$$\varphi_K^2 = \int_K \left[a\left(\widetilde{\nabla u} - \nabla u_h\right)^2 + b(\tilde{u} - u_h)^2 \right] dx$$

where $\widetilde{\nabla u}$ and $\tilde{u}$ are the postprocessed values of the gradient ∇u_h and solution u_h.

There are a number of ways of calculating the postprocessed values of u_h, perhaps the best known being the use of *extraction formulas*, discussed in reference [7]. Here we will only mention that good postprocessing is computationally expensive, and error estimators based on postprocessing are used for final verification of the quality of the results, rather than as error indicators for mesh refinement.

h-p Adaptive Mesh Refinement

The error estimators discussed in this section may be used as a basis for adaptive mesh refinement strategy. In the case of *h-p* meshes, the optimal mesh refinement, even with properly calculated error indicators, is a difficult problem. The primary question is: *h*-refinement or *p*-enrichment? Additional difficulties also result from the unrefinement option, the selection of the version of *p*-refinement, etc.

In this section we will present a summary of an *h-p* adaptive refinement strategy developed recently at COMCO. A more detailed discussion, including the theoretical background, can be found in reference [61].

We begin by introducing the distribution of mesh sizes and spectral orders over Ω_h, characterized by functions h and p:

$$\left.\begin{aligned} h(x_1,x_2) &= h_K \quad (x_1,x_2) \in K \\ p(x_1,x_2) &= p_K \quad (x_1,x_2) \in K \end{aligned}\right\}$$

the *error indicator* θ_K for element K will depend on h_K and p_K and is represented by a local error density $\varphi_K = \varphi_K(h,p) : \theta_K = \int_K \varphi_K(h,p)dx$. Thus, an indication of the total error over Ω is given by the functional

$$J(h,p) = \sum_{K \in \Omega_h} \int_K \varphi_K(h,p)dx$$

Next, let n_K denote the total number of degrees of freedom of element K. We introduce a *degree-of-freedom density* $n = n(h,p)$ so that the total number of degrees of freedom M is given by

$$\left.\begin{aligned} M &= \int_\Omega n(h,p)dx \\ n(h,p)|_K &= n_K \end{aligned}\right\}$$

The optimal mesh for a fixed number $M = M_0$ degrees of freedom will be defined by the h,p-distribution (h^*,p^*) such that

$$J(h^*,p^*) = \min_{\substack{\int_\Omega n(h,p)dx \\ = M_0}} J(h,p)$$

To resolve this problem using the method of Lagrange multipliers, we introduce the Lagrangian,

$$L(h,p;\lambda) = J(h,p) - \lambda\left(\int_\Omega n(h,p)dx - M_0\right).$$

and arrive at the optimality conditions

$$\left.\begin{aligned} \left.\frac{d\varphi_K}{dn}\right|_{p=\text{const.}} &= \lambda = \text{const.} \\ \left.\frac{d\varphi_K}{dn}\right|_{h=\text{const.}} &= \lambda = \text{const.} \end{aligned}\right\}$$

In practical applications these principles are implemented in the form of a simple refinement strategy: *perform refinement for which the anticipated decrease of the error per new degree of freedom are as large as possible.*

According to this principle, the algorithm for h-p refinements is as follows:

1. For all elements in an initial mesh, compute the anticipated decreases of errors for all of the refinements which may actually occur.
2. For every element evaluate $\Delta\theta_p/\Delta n_p, \Delta\theta_h/\Delta n_h$. (For interpolation errors, use the procedure outlined below.)

   Set $\Delta\theta/\Delta n = \max(\Delta\theta_p/\Delta n_p, \Delta\theta_h/\Delta n_h)$ and store the larger of these two quantities.
3. Scan the mesh and identify the largest value of $\Delta\theta/\Delta n$ in the mesh: $(\Delta\theta/\Delta n)_{\max}$.
4. Create a list of elements for which $\Delta\theta/\Delta n \geq \alpha_1 \cdot (\Delta\theta/\Delta n)_{\max}$.
5. Perform refinements of elements on the list. The type of refinements corresponds to information stored in item 2.
6. Update the solution: solve the problem on the new mesh.
7. Estimate the global error $J = \sum_k \theta_k$. If $J \leq \alpha_2$, stop, otherwise go to 1.

The calculation of $\Delta\theta_p/\Delta n$ and $\Delta\theta_h/\Delta n$ for interpolation error indicators may be obtained using the following procedure:

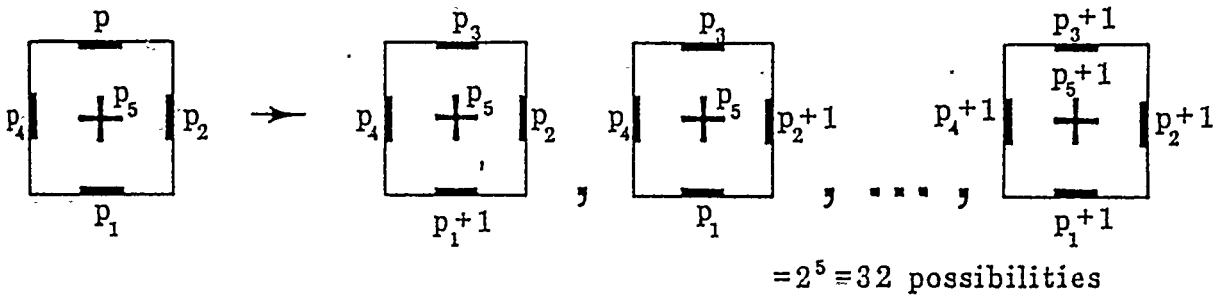


Figure 6.5: Possible ways of enriching an element.

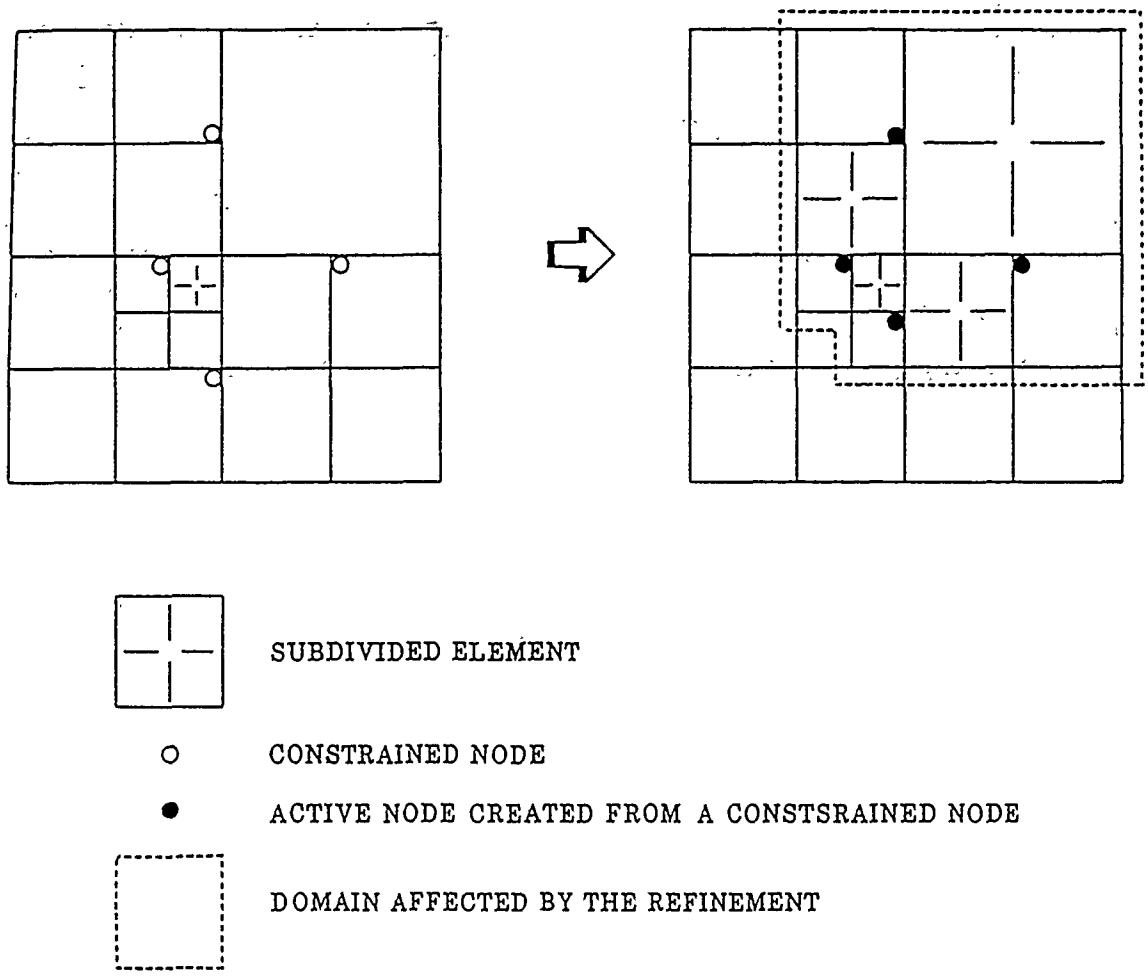
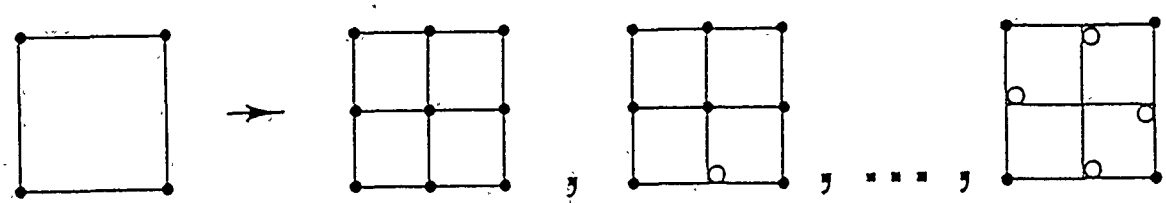


Figure 6.6: Modification of the mesh caused by subdivision of a constrained element.

(a) Subdivisions:



○ -Constrained Node

● -Active Node

(b) Removing constrained nodes:

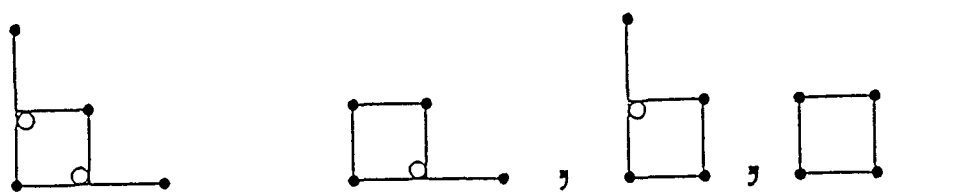


Figure 6.7: Possible ways of subdividing an element and removing the constrained nodes.

2.1 Subroutine for determining $\Delta\theta_p/\Delta n_p$.

(a) Evaluate new orders of nodes of a given element K_i and its neighbors.

(b) Compute $\Delta\theta_p$ using errors for K_i and its neighbors in the situations shown in Fig. 6.5.

(c) Determine the number of the new degrees of freedom Δn_p and compute $\Delta\theta_p/\Delta n_p$.

2.2 Subroutine for determining $\Delta\theta_h/\Delta n_h$:

(a) Evaluate the sequence of elements that must be subdivided due to refining the given element K_i and the "two-to-one" rule (see Fig. 6.6).

(b) Determine elements with disappearing constrained nodes (Fig. 6.6).

(c) Compute $\Delta\theta_h$ using errors for all the subdividing elements found in (a) and elements listed in (b); appropriate situations from Fig. 6.7 must be considered.

(d) Count the number of new degrees of freedom Δn_h and compute $\Delta\theta_h/\Delta n_h$.

In practical applications, this algorithm can be augmented by some rather heuristic rules, such as:

- In boundary layers in viscous flows the p-refinement is recommended with a possible anisotropic distribution of p (higher p in the direction normal to the wall).
- In regions with low regularity of the solution, such as shocks, the polynomial level should be kept rather low and h-refinement is preferred.

Rules of this type can be effectively implemented in the expert system, which will be combined with the above algorithmic procedure to produce strategies of maximum robustness and efficiency.

6.2.2 Adaptive Timestepping Techniques

Time–dependent problems are usually solved by time marching techniques, where the time domain is discretized into a number of steps $t_1, t_2, \ldots, t_n$, and the solution proceeds by the solution of a sequence of incremental problems. In the simplest algorithms the time step interval is kept constant throughout the whole computational process. The actual value of the time step Δt is estimated from stability requirements, accuracy expectations, or other heuristic estimates. In most practical applications, a fixed time step leads to relatively inefficient solutions and adaptive adjustements of Δt are very desirable.

In this section we will present an adaptive time-stepping technique implemented for thermo-viscoplastic structural analysis governed by the Bodner-Partom constitutive equations. The details of this formulation can be found in reference [74]. Here we will focus on the issue of adaptive time-stepping.

The variable time step algorithm is a modified Euler scheme using a truncation error criterion to adjust the time step.

For simplicity, consider the single ordinary differential equation,

$$\dot{y} = f(y,t) \tag{6.9}$$

The solution is advanced using a predictor-corrector scheme. The predictor phase consists of an Euler step:

$$y^P_{t+\Delta t} = y_t + \Delta t \dot{y}_t \tag{6.10}$$

$$\dot{y}^P_{t+\Delta t} = f\left(y^P_{t+\Delta t}, t+\Delta t\right) \tag{6.11}$$

An error indicator E is then computed from

$$E = \frac{\left|\Delta t\left(\dot{y}^P_{t+\Delta t} - \dot{y}_t\right)\right|}{2\left|y^P_{t+\Delta t}\right|} \tag{6.12}$$

The error indicator is next compared with a preset error criterion and if the criterion is met, the time step is sufficiently small enough to proceed to the corrector stage. Otherwise, the predictor phase for Eqs. (6.10)-(6.11) is repeated with a smaller time step. For the Bodner-Partom evolution equations the control variables used to calculate the error indicator were the components of a stress tensor σ_{ij}, state variables Z_i, and plastic work W_p, with the maximum of these selected as the error.

The corrector phase is the modified Newton scheme,

$$\dot{y}_{avg} = \left(\dot{y}_t + \dot{y}^P_{t+\Delta t}\right)/2$$

$$\dot{y}^C_{t+\Delta t} = y_t + \Delta t \dot{y}_{avg}$$

A flowchart depicting the adaptive scheme is shown in Fig. 6.8. The flowchart shows how the time step is either reduced or increased depending on the error indicator, Eq. (6.12). The flowchart shows that the time step is reduced or increased by a factor of two. This approach is effective, but an alternate scheme also has been used where the new time step is based on the error E. In this scheme

$$\Delta t = \Delta t \sqrt{\frac{E_{opt}}{E}}$$

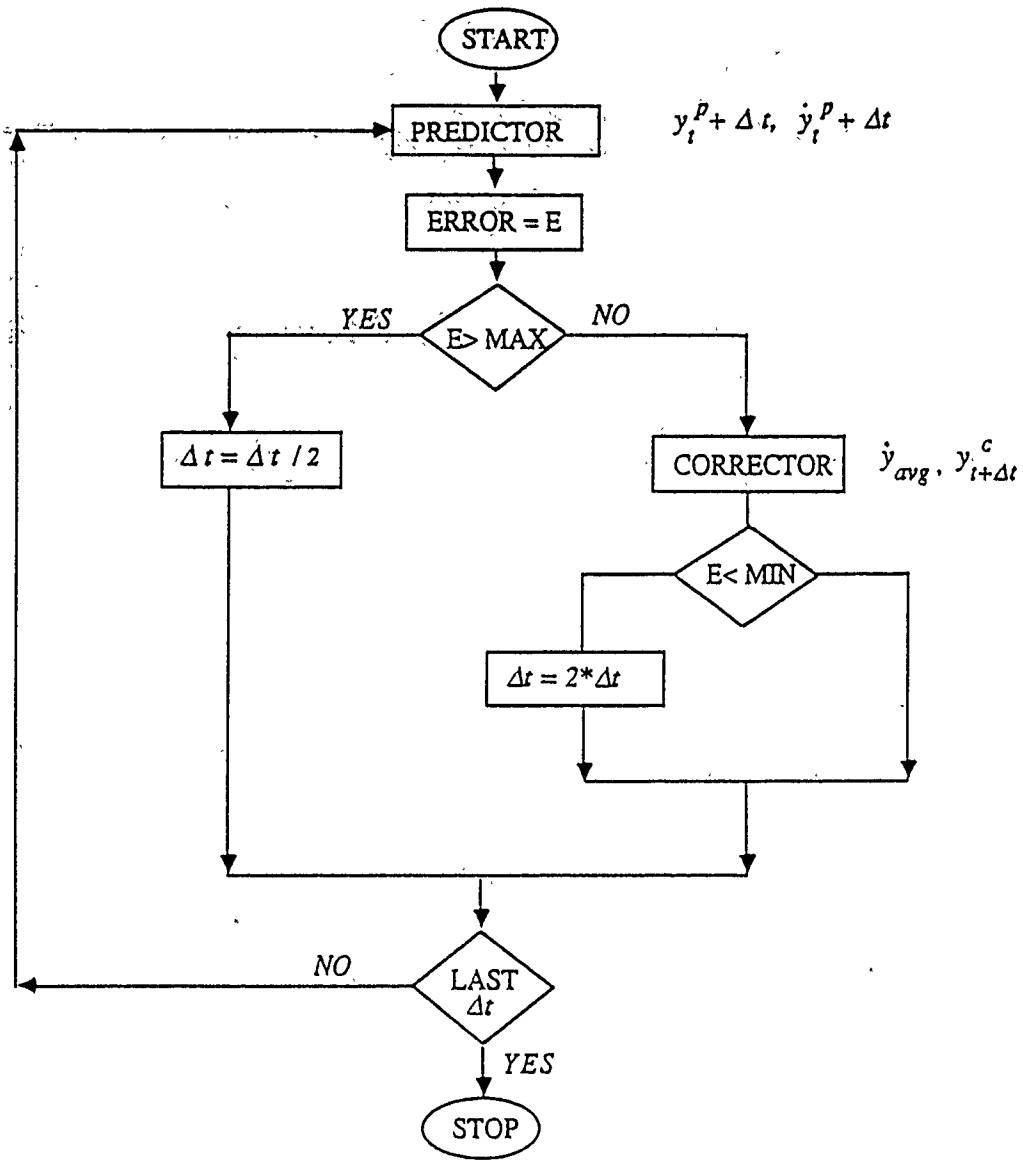


Figure 6.8: Adaptive timestepping algorithm for viscoelastoplastic evolution problems.

where $E_{opt} = \sqrt{E_{max} E_{min}}$. Error tolerances E_{max} and E_{min} are specified by the user, typical values being $E_{max} = 0.05$, $E_{min} = 0.0005$. In practical implementations, it is possible to use an expert system to adjust the time step in an even more efficient fashion.

The effectivity of an adaptive timestepping algorithm is clearly illustrated by the example of a viscoplastic bar subjected to thermal loads. The details of this example are discussed elsewhere [74]. Here we will present only the calculated evolution of temperature and stress (Fig. 6.9a and b) and focus on the evolution of the adaptive time step and comparison with a fixed time step algorithm. In the fixed time step algorithm, 1200 time steps of size 0.001 sec. were used. The variable time step algorithm, shown in Fig. 6.8, was then implemented and the problem was resolved. Figure 6.9 shows the history of the variable time step and indicates that the new analysis required only 213 steps—a substantial savings. The figure shows that in the "flat" part of the stress response a large time step was used, but near $t = 0$ and again at $t = 0.5$ sec. when the stress is changing rapidly, small time steps are needed to capture the response accurately.

6.2.3 Performance Monitoring and Control of Computational Procedures

Computational procedures used in advanced numerical analysis usually exhibit a different performance and reliability level for different problem classes. This is true primarily for advanced, time–dependent and nonlinear problems, where characteristics of the solution, stability, and conditioning strongly affect the performance of the method.

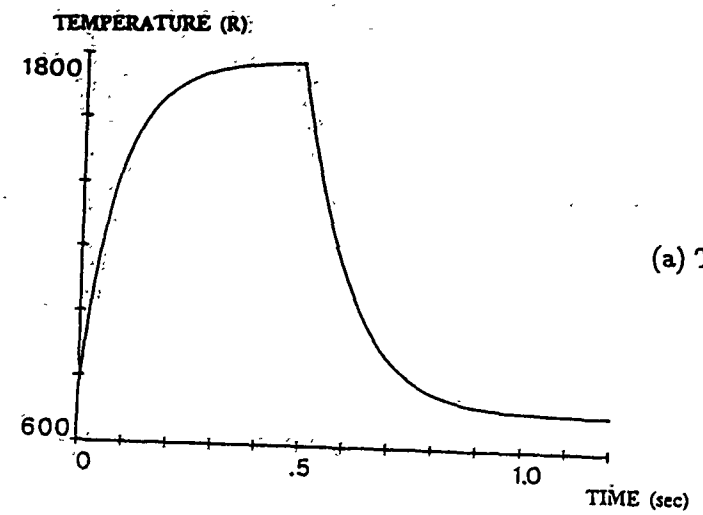
It is not unusual for the analyst to have to closely watch the performance of various methods, adjust parameters during the solution process, restart the solution or even repeat the computations with a different selection of parameters. It appears that automation of these processes should:

- improve the performance of advanced codes when used by inexperienced users, and
- release experienced analysts from the responsibility of closely monitoring the numerical processes and adjusting the parameters.

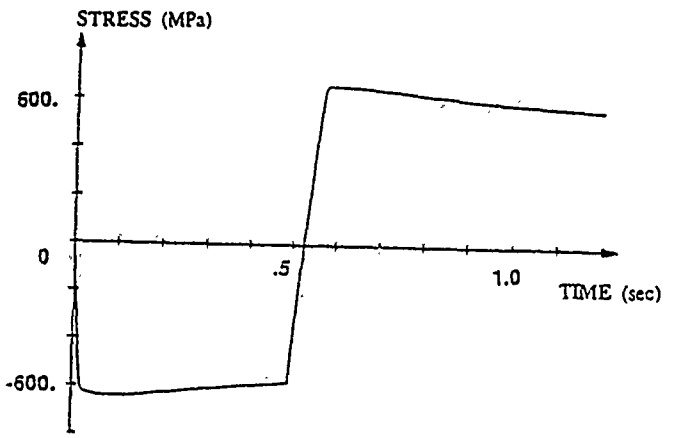
In this section we focus on the automation of rather heuristic tasks, namely the monitoring of the convergence of certain iterative methods. As a particular example we consider nonlinear algorithms, based on Newton's method, which are notorious for poor convergence in many practical problems and for high sensitivity to various parameters of the system. For these problems an automated approach was developed and implemented which simulates the performance of an experienced analyst in monitoring the convergence of these methods.

Consider a quasi–static nonlinear problem defined by the system of algebraic equations:

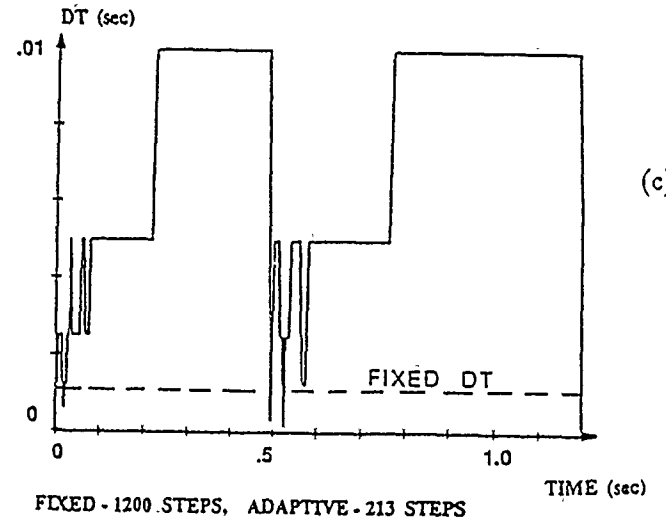
$$L(u) = F(t)$$



(a) Temperature history



(b) Stress history



(c) Adaptive time step history

Figure 6.9: Adaptive timestepping in thermo–viscoplastic analysis: (a) temperature history, (b) stress history, and (c) time step history.

where u is the solution vector, t is a load parameter (quasi-static time), and L and R are left hand and right hand vectors, respectively. This type of equation is obtained from the finite element discretization of nonlinear boundary-value problems. The problem is usually solved by considering a sequence of load parameters $t^1, t^2, \ldots, t^n$. For each new load the nonlinear problem is solved by an iterative procedure:

$$K(u_i)\delta u_{(i+1)} = F(t^n) - L(u_i)$$
$$u_{(i+1)} = u_i + \delta u_{(i+1)}$$

In the automation of this process, an expert system can be used to adjust the solution sequence, the load parameter, and the number of iterations according to the performance of the method.

Assume that at time step t^n the code performed m iterations, where m is the lesser of the user-prescribed limit and the number necessary for convergence. The error values (in an appropriate norm) at consecutive iterations are $e_1, e_2, \ldots, e_m$. If the error e_m is below the prescribed tolerance, the process is considered to be converged and the code proceeds to the next load value. The expert system will adjust the load step according to a simple heuristic formula:

$$\Delta t_{(n+1)} = \Delta t \ (m_{opt}/m)$$

where m_{opt} is an optimum number of iterations per time step (usually 5 or 6). The difficulties arise when the iterative process is not convergent. Typical existing implementations will terminate the computations in this case or, even worse, continue computations to produce nonconverged, useless answers. The automated approach developed in this project is based on the application of an expert system to make decisions regarding the continuation of the computations.

A starting point for the expert system is an analysis of trends in the behavior of the error at consecutive iterations. First, the error history is converted to a logarithmic scale, so that error histories corresponding to a power-type convergence appear as straight lines with different slopes. Then a curve fitting based on a second-order weighted minimization method [45] is performed to estimate the behavior of the error (see Fig. 6.10). The weight factor used here is the largest for the last iteration e_m and diminishes for preceding iterations according to the formula:

$$w_i = m_{opt}/(m_{opt} + m - i)$$

The behavior of the process is estimated by monitoring the slope of the error curve at the last iteration: $(de/di)_{(i=m)}$. According to this behavior, decisions are being made concerning the iterative process. The details of the corresponding rules can be found in the listing of the expert system in Appendix C.1. Here it suffices to note that:

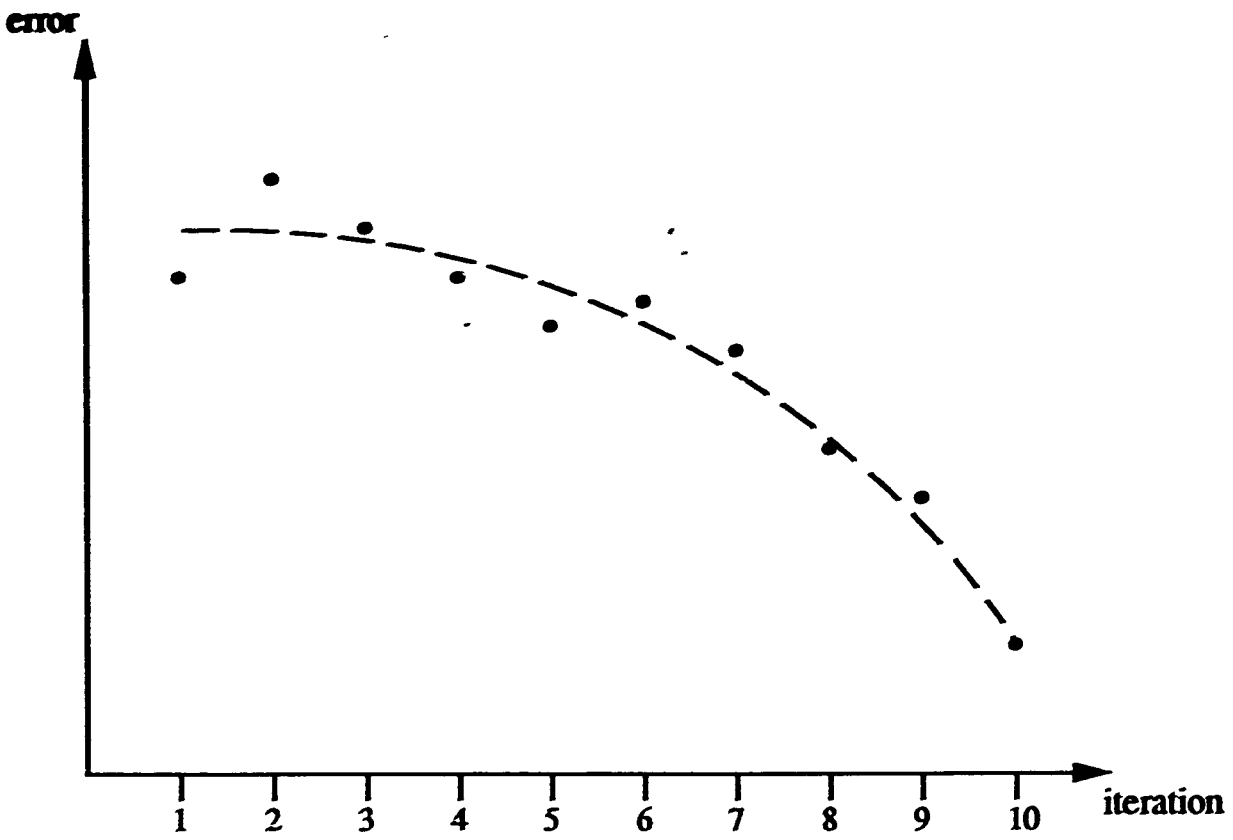


Figure 6.10: Curve fitting for error history in nonlinear problems.

1. If the slope is negative (diminishing error), then the expert system decides to continue itrations. However, if this situation is repeated several times with no convergence, the case is classified as hopeless and treated as divergent. A warning is issued to the user.

2. If the slope is close to zero (the method stagnates) or positive (the method diverges), then the results of the iterative process are discarded, the solution backs off to the last step, and the load increment is reduced by a factor of two. If this procedure repeats several times with no success, the whole process is terminated and a warning is issued to the user with identification of probable reasons of nonconvergence and possible remedies.

The above description outlines basic ideas of monitoring the performance of nonlinear processes. The actual expert system has several additional rules. Some of them check for divergence caused by problems in other parts of the code (say poor conditioning in the frontal solver), others verify non-reasonable combinations of user defined parameters, etc. Details can be found in the listing of the expert system in Appendix C.2.

This relatively simple procedure has been found to be very effective in practical applications and has provided essential savings of both computational time and the time of human analysts. Several examples are discussed in Section 8. The above approach can also be used for monitoring the performance of other iterative techniqus, such as iterative solvers for linear systems of equations, Riks method, etc.

6.2.4 Adaptive Selection of Implicit and Explicit Time Integration Schemes

One of the recently explored types of adaptive techniques is an adaptive selection of implicit and explicit time integration schemes. The basic idea is to combine the robust, unconditionally stable implicit schemes with the relatively inexpensive explicit schemes to achieve the maximum effectiveness at the minimum computational cost.

Such a fully adaptive implicit/explicit techniqe was developed at COMCO for the solution of compressible viscous flows. These flows are governed by Navier-Stokes equations [54,55,79] of the form:

$$\dot{\boldsymbol{u}} + \boldsymbol{F}^C_{i,i} = \boldsymbol{F}^V_{i,i}$$

where $\boldsymbol{u}$ is the vector of conservation variables, $\dot{\boldsymbol{u}} = \partial \boldsymbol{u}/\partial t, \boldsymbol{F}^C_k$ are inviscid (convective) flux vectors, and $\boldsymbol{F}^V_i$ are viscous flux vectors. Indices i in the above formula refer to axes of a Cartesian coordinate system, a comma denotes partial differentiation and the summation convention is applied. The components of vectors $\boldsymbol{u}$, $\boldsymbol{F}^C_i$, and $\boldsymbol{F}^V_i$ are given in two-dimensional case by:

$$\boldsymbol{u} = \{\rho, \rho v_1, \rho v_2, \rho e\}^T = \{\rho, m_1, m_2, E\}^T$$

$$F_i^C = \begin{bmatrix} \rho v_i \\ \rho v_1 v_i & + & p\delta_{1i} \\ \rho v_2 v_i & + & p\delta_{2i} \\ (E+p)v_i \end{bmatrix}$$

$$F_i^V = \begin{bmatrix} 0 \\ \sigma_{1i} \\ \sigma_{2i} \\ v_m \sigma_{mi} & + & q_i \end{bmatrix}$$

where ρ is the fluid density, p is the pressure, v_i are velocity components, e is a total energy per unit mass, σ_{ij} are stress components, and q_i are components of a heat flux vector.

Convective fluxes are functions of the conservation variables only:

$$F_i^C = F_i^C(u)$$

so that inviscid Jacobians are defined by:

$$A_i = \frac{\partial F_i^C}{\partial u}$$

Viscous fluxes depend on both conservation variables and their gradients:

$$F_i^V = F_i^V(u, u_j)$$

and the corresponding Jacobians are defined by

$$P_i = \frac{\partial F_i^V}{\partial u}$$

$$R_{ij} = \frac{\partial F_i^V}{\partial u_j}$$

For these equations, a general family of implicit one-step methods was developed [79] leading to the incremental equation to be solved at each time step:

$$\begin{aligned} \Delta u &+ \alpha \Delta t \left(A_i^n \Delta u\right)_{,i} - \gamma \Delta t \left[\left(R_{ij}^n \Delta u_j\right)_{,i} + \left(P_i^n \Delta u\right)_{,i}\right] + \\ &- (1-2\alpha)\beta \frac{\Delta t^2}{2} \left(A_i^n A_j^n \Delta u_j\right)_{,i} \\ &= \Delta t \left(F_{i,i}^{Vn} - F_{i,i}^{Cn}\right) + (1-2\alpha)\frac{\Delta t^2}{2} \left(A_i^n F_{k,k}^{Cn}\right)_{,i} \\ &+ (\alpha - \gamma) O(\mu, k) O\left(\Delta t^2\right) + (1-2\alpha) O(\mu, k) O\left(\Delta t^2\right) + O\left(\Delta t^3\right) \end{aligned} \tag{6.13}$$

The above equation is combined with the appropriate boundary conditions to form a boundary–value problem over the domain Ω. The incremental solution at consecutive time steps are accumulated to formulate a solution of the time–dependent problem.

Formula (6.13) is a linear equation for the increment Δu. The formulation in general is implicit, except when $\alpha = \beta = \gamma = 0$, in which case the explicit scheme is recovered. The interpretation of different implicitness parameters is as follows:

α — represents implicitness of the advective terms,

β — represents implicitness of the viscous terms,

γ — represents implicitness of the second order terms.

The variational formulation of the problem is obtained by introducing the space of test functions:

$$V = \{\boldsymbol{v} = (v_1, v_2 \ldots v_M) \; s.t. \; v_i \in H^1(\Omega) \text{ and } v_i = 0 \text{ on } \Gamma_{Di}\} \tag{6.14}$$

where M is the number of conservation variables, $H^1(\Omega)$ is the usual Sobolev space of functions with derivatives in $L^2(\Omega)$, and Γ_{Di} is the boundary with specified Dirichlet boundary conditions for variable u_i. After multiplication of the incremental equation (6.13) by an arbitrary test function $\boldsymbol{v}(\boldsymbol{x}) \in V$, integrating over the domain Ω and application of the divergence theorem, the following weak formulation of the problem is obtained:

Find $\Delta \boldsymbol{u} \in V s.t. \; \forall \, \boldsymbol{v} \in V$:

$$\begin{aligned}
&\int_\Omega (\Delta \boldsymbol{u} \cdot \boldsymbol{v} - \alpha \Delta t \boldsymbol{A}_i^n \Delta \boldsymbol{u} \cdot \boldsymbol{v}_{,i} + \gamma \Delta t \boldsymbol{R}_{ij}^n \Delta u_j \cdot \boldsymbol{v}_{,i} \\
&+ \gamma \Delta t \boldsymbol{P}_i^n \Delta \boldsymbol{u} \cdot \boldsymbol{v}_{,i} + (1 - 2\alpha)\beta \frac{\Delta t^2}{2} \boldsymbol{A}_i^n \boldsymbol{A}_j^n \Delta \boldsymbol{u}_{,j} \cdot \boldsymbol{v}_{,i}) d\Omega \\
&+ \int_{\partial\Omega} (\alpha \Delta t n_i \boldsymbol{A}_i^n \Delta \boldsymbol{u} \cdot \boldsymbol{v} - \gamma \Delta t n_i \boldsymbol{R}_{ij} \Delta \boldsymbol{u}_{,j} \cdot \boldsymbol{v} \\
&- \gamma \Delta t n_i \boldsymbol{P}_i \Delta \boldsymbol{u} \cdot \boldsymbol{v} - (1 - 2\alpha)\beta \frac{\Delta t^2}{2} n_i \boldsymbol{A}_i^n \boldsymbol{A}_j^n \Delta \boldsymbol{u}_{,j} \cdot \boldsymbol{v}) dS \\
&= \int_\Omega (\Delta t \left(\boldsymbol{F}_i^{Cn} - \boldsymbol{F}_i^{Vn} \right) \cdot \boldsymbol{v}_{,i} - (1 - 2\alpha) \frac{\Delta t^2}{2} \boldsymbol{A}_i^n \boldsymbol{F}_{j,j}^{Cn} \cdot \boldsymbol{v}_{,i}) d\Omega \\
&+ \int_{\partial\Omega} (-\Delta t n_i \left(\boldsymbol{F}_i^{Cn} - \boldsymbol{F}_i^{Vn} \right) \cdot \boldsymbol{v} + (1 - 2\alpha) \frac{\Delta t^2}{2} n_i \boldsymbol{A}_i^n \boldsymbol{F}_{j,j}^{Cn} \cdot \boldsymbol{v}) dS
\end{aligned} \tag{6.15}$$

The above variational equation is the basis for a standard finite element discretization of the problem. The implicit/explicit approach is introduced by partitioning the computational

domain into two subregions $\Omega^{(I)}$ and $\Omega^{(E)}$ such that:

$$\Omega^{(E)} \cap \Omega^{(I)} = \Gamma_{EI} \ , \ \Omega^{(E)} \cup \Omega^{(I)} = \Omega$$

It is convenient to assume that the interface between the two regions coincides with the element boundaries.

It can easily be observed that the differential equations to be solved on the two subregions are different due to different implicitness parameters applied in each zone: $\alpha^{(I)}, \beta^{(I)}, \gamma^{(I)}$ in the implicit zone and $\alpha^{(E)}, \beta^{(E)}, \gamma^{(E)} = 0$ in the explicit zone . Therefore, the variational formulation (6.15), based on the assumption of constant implicitness parameters, cannot be applied to the domain Ω. Instead, it must be applied separately to each subdomain with additional continuation conditions across the interface. These conditions represent continuity of the solution and satisfaction of the conservation laws across the interface and are of the form:

$$\left.\begin{aligned} \boldsymbol{u}^{(E)} &= \boldsymbol{u}^{(I)} \\ \boldsymbol{F}_i^{(E)C} &= \boldsymbol{F}_i^{(I)C} \\ \boldsymbol{A}_i^{(E)} &= \boldsymbol{A}_i^{(I)} \\ \boldsymbol{F}_n^{(E)V} &= \boldsymbol{F}_n^{(I)V} \end{aligned}\right\} \text{ on } \Gamma_{EI} \tag{6.16}$$

where index n refers to the outward normal for the corresonding region ($\boldsymbol{n}^{(E)} = -\boldsymbol{n}^{(I)}$). The continuity requirement also pertains to the test function, so that $\boldsymbol{v}^{(E)} = \boldsymbol{v}^{(I)} = \boldsymbol{v}$. Note that for general weak solutions of Euler equations the solution $\boldsymbol{u}$ need not be continuous across the interface. However, for regularized problems and finite element interpolation, the continuity of $\boldsymbol{u}$ is actually satisfied.

If the variational statement is formulated for this problem, then in addition to interior integrals for each subdomain and regular boundary integrals, jump integrals across the interface appear in the formulation. These additional interface terms appearing on the right–hand side are of the form:

$$\int_{\Gamma_{IE}} \Big\{ -\Delta t \left[\boldsymbol{F}_n^{(I)Cn} + \boldsymbol{F}_n^{(E)Cn} \right] \cdot \boldsymbol{v} + \Delta t \left[\boldsymbol{F}_n^{(I)Vn} + \boldsymbol{F}_n^{(E)Vn} \right] \cdot \boldsymbol{v}$$

$$+ \frac{\Delta t^2}{2} \left[\left(1 - 2\alpha^{(I)} \right) \boldsymbol{A}_n^{(I)} \boldsymbol{F}_{j,j}^{(I)Cn} + \left(1 - 2\alpha^{(E)} \right) \boldsymbol{A}_n^{(E)} \boldsymbol{F}_{j,j}^{(E)Cn} \right] \cdot \boldsymbol{v} \Big\} \, ds$$

On the left–hand side of the variational formulation additional interface terms are of the form:

$$\int_{\Gamma_{IE}} \Big\{ \Delta t \left[\alpha^{(I)} \Delta \boldsymbol{F}_n^{(I)C} + \alpha^{(E)} \Delta \boldsymbol{F}_n^{(E)C} \right] \cdot \boldsymbol{v} - \Delta T \left[\gamma^{(I)} \Delta \boldsymbol{F}_n^{(I)V} - \gamma^{(E)} \Delta \boldsymbol{F}_n^{(E)V} \right] \cdot \boldsymbol{v}$$

$$- \frac{\Delta t^2}{2} \left[(1 - 2\alpha^{(I)}) \beta^{(I)} \boldsymbol{A}_n^{(I)} \Delta \boldsymbol{F}_{j,j}^{(I)C} + (1 - 2\alpha^{(E)}) \beta^{(E)} \boldsymbol{A}_n^{(E)} \Delta \boldsymbol{F}_{j,j}^{(E)C} \right] \cdot \boldsymbol{v} \Big\} \, ds$$

In order to enforce interface conditions, the values of consecutive terms in the above formulas should be prescribed using equation (6.16).

An important issue for the automation of implicit/explicit procedures is the actual selection of implicit/explicit zones. The basic criterion is simple: for a given time step all nodes which violate the stability criterion for the explicit scheme should be treated with the implicit scheme. According to this criterion, several options for an automatic adaptive selection of implicit/explicit zones were implemented, such as a user–prescribed time step, a user–prescribed CFL number, or a prescribed percentage of implicit elements. These relatively simple options are described in reference [79]. Here we will focus on the "smartest" fully automatic version, based on the minimization of the cost of computations.

In this option, the time step and the implicit/explicit subzones are selected to minimize the cost of advancing the solution in time (say one time unit). The algorithm is based on the fact that, for an increased time step, an increasing number of elements must be analyzed with the (expensive) implicit algorithm. The typical situation is presented in Fig. 6.11, which shows for different time steps the relative number of nodes that must be treated with the implicit scheme (to preserve stability). On the abscissa, the Δt_{FE} denotes the longest time step allowable for the fully explicit scheme (with certain safety factors). Δt_{FI} denotes the shortest time step requiring a fully implicit procedure. The relative number of implicit nodes increases as a step function from zero for $\Delta t \leq \Delta t_{FE}$ to one for $\Delta t \geq t_{FI}$. Now assume that the ratio r of the computational cost of processing one implicit node to the cost of processing one explicit node is given. This ratio can be estimated relatively well by comparing the calculation time of element matrices and adding, for implicit nodes, a correction for the solution of the system of equations. Then the reduction of the cost of advancing the solution in time with the implicit/explicit scheme, as compared to the fully explicit scheme, is given by the formula:

$$R(\Delta t) = \frac{\Delta t_{FE}}{\Delta t}\left(n^{(E)} + rn^{(I)}\right)$$

Typical plots of the function $R(\Delta t)$ are presented in Fig. 6.11. Shown here are the two cases: the case of a small difference between fully explicit and fully implicit time steps—an almost uniform mesh the case of a large difference between fully explicit and fully implicit time steps. Note that in either case, restrictions on the length of the time step should be applied, for example, from the maximum CFL condition. Otherwise the cheapest procedure would always be to reach the final time with one implicit step.

From the plots in Fig. 6.11, the following observation can be made: for an essentially uniform mesh, the mixed implicit/explicit procedure does not provide savings of the computational cost—either a fully implicit or fully explicit scheme is the cheapest depending on the time step restriction. On the other hand, for very diverse mesh sizes the mixed

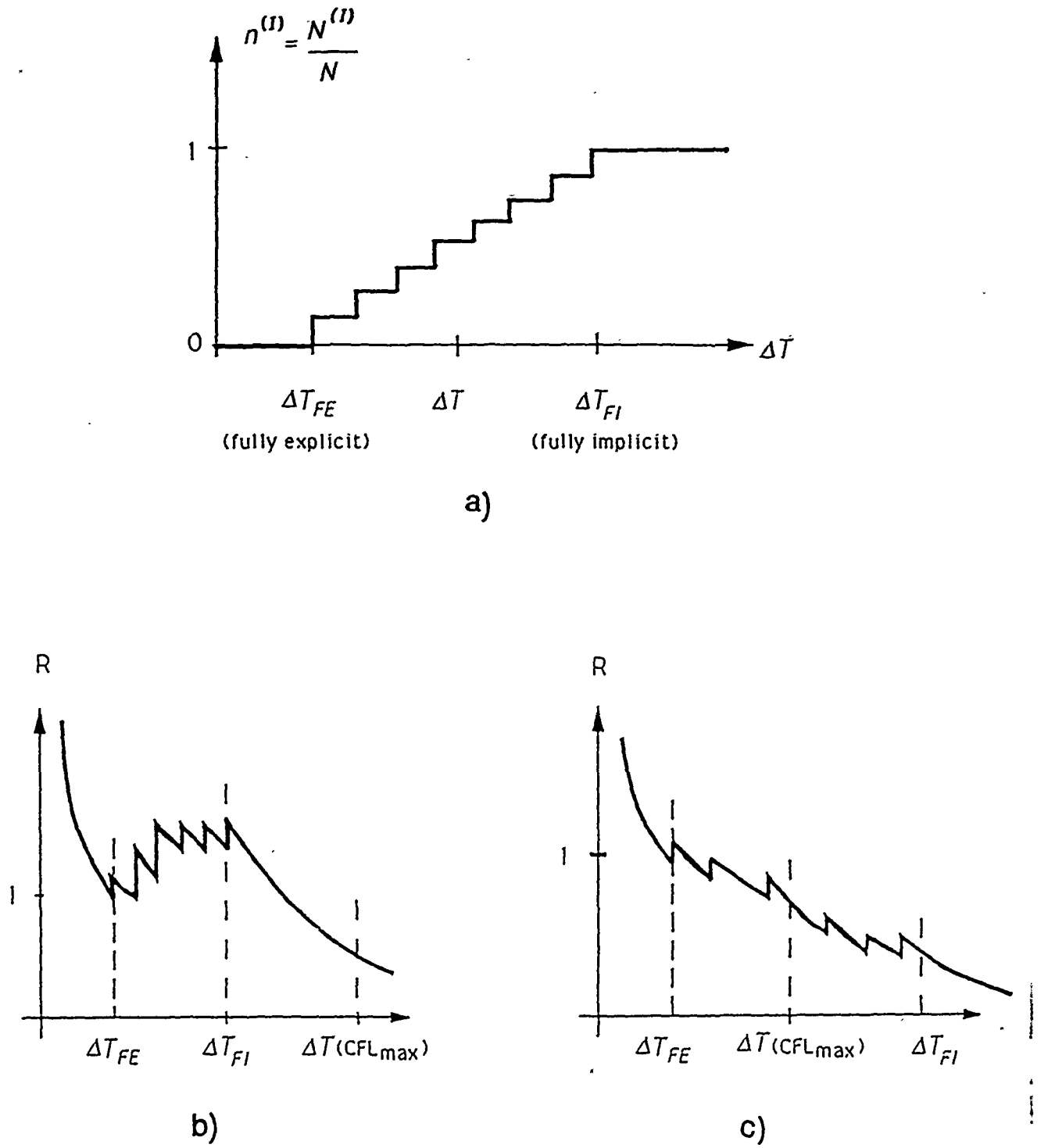


Figure 6.11: Reduction of the cost of computations due to implicit/explicit procedure.

procedure provides considerable savings. This means that the effectiveness of the mixed implicit/explicit scheme will be the best for large-scale computations with both very large and very small elements present in the domain. In the practical implementation of this method, the approximation of the function $R(\Delta t)$ is automatically estimated for a given mesh. Then, the time step corresponding to the smallest $R(\Delta t)$ is selected automatically (subject to additional constraints, in particular the $\mathrm{CFL}_{\max}$ constraint).

The above implicit/explicit procedure results in a system of equations

$$\left(\boldsymbol{M} + \boldsymbol{K}^{(I)}\right) \boldsymbol{\Delta U} = \boldsymbol{R}$$

where the stiffness matrix $\boldsymbol{K}$ has non–zero entries only for degrees of freedom in the implicit zone or for nodes with penalty–enforced boundary conditions. The cost minimization option has been implemented in the adaptive finite element code. Selected numerical examples of automated implicit/explicit procedures will be presented in Section 8.

It should be noted here that in addition to the above criteria, based purely on stability analysis, some other criteria for application of implicit schemes can be applied. For example, within boundary layers the implicit scheme may be preferred, because it provides faster convergence of the boundary fluxes. Even more complicated issues arise in boundary layer–shock interaction problems, when adaptive implicit/explicit schemes need to be combined with adaptive application of artificial dissipation, etc. At the current state of knowledge in the computational fluid dynamics rules related to these problems are rather heuristic and can best be handled by an appropriate expert system. Therefore the ultimate adaptive implicit/explicit solver should combine smart algorithms with heuristic knowledge represented by expert systems and other knowledge engineering techniques.

6.3 Generalized Postprocessing

The final solution of the finite element method is often obtained in the form of tables of nodal values of the basic unknowns of the problems, e.g., displacements in the solid mechanics problems. This solution is usually postprocessed in order to present the results in a more comprehensive form or to calculate new functions of interest to the user. Typical functions of postprocessing modules are:

1. calculation of auxiliary functions (strains, stresses) derived from the primary solution,
2. use of special postprocessing techniques to obtain results of higher accuracy than the primary solution,
3. estimation of solution errors, and

4. presentation of results in the form of plots, diagrams, tables, etc.

The above tasks are basically algorithmic. However, advanced programs usually offer several possibilities for performing these tasks and the selection of the method is expected from the user. This is especially true for items two and three, since a variety of algorithms, each of different quality and computational cost, is available for advanced postprocessing or error estimation (see references [7,8,9,15,16,60,84]).

The postprocessing stage is important in the design process for yet another reason: it provides information necessary for the acceptance or rejection of the solution and for modification of the input data in order to obtain a satisfactory solution.

A fully automated version of a design process should allow the knowledge–based system to make decisions concerning the acceptability of the results and to modify the model if necessary. The question then arises as to the best ways to extract essential information about the solution in a form acceptable to the knowledge–based system. In today's finite element codes, oriented toward interaction with the user, a graphical display of results is preferred. This graphical information is, however, unavailable to the expert system. Instead, a limited number of compact pieces of information should be produced, possibly in the form of numbers or short statements.

In this section we present studies related to this problem. First we study methods of extracting essential characteristics of a generic function $f(x)$, in particular the information related to its distribution within the domain Ω. Then we apply these considerations to typical parameters of computational solid mechanics, such as displacements, rotations, strains, stresses, etc.

6.3.1 Essential Characteristics of an Arbitrary Function

Suppose that within a certain domain Ω defined is a function $f(\boldsymbol{x})$: such that $f : \Omega \rightarrow R$. At this point we will not specify any special assumptions about the regularity of f.

The most natural parameters characterizing the function f include its minimum, maximum and average values, defined as:

$$f_{\max} = \max_{\boldsymbol{x}\in\Omega} f(\boldsymbol{x})$$

$$f_{\min} = \min_{\boldsymbol{x}\in\Omega} f(\boldsymbol{x})$$

$$f_{\text{ave}} = \frac{\int_\Omega f(x)dx}{\int_\Omega dx}$$

Other parameters of interest include the locations $x_{f\min}$, $x_{f\max}$ corresponding to minimum and maximum values, respectively. Moreover, it is often useful to know the percentage of the domain, where f is greater than a prescribed threshold value f_T. This area is defined as:

$$A = \frac{\int_{\overline{\Omega}} dx}{\int_{\Omega} dx}$$

where $\overline{\Omega} = \{x,\ s.t.\ f(x) \geq f_T\}$. The above parameters can easily be extracted from the finite element data.

A more complex problem arises if one wants to represent a distribution of the function within the domain Ω, for example to detect concentrations of high values in certain regions, periodic nature of f, etc. For periodic and quasi-random functions, spectral analysis may be used to characterize the distribution of $f(x)$. For detecting other special characteristics of f the following procedure is proposed:

First, define a class of functions

$$T = \{t(x)\ ,\ s.t.\ t(x) \text{ exhibits behavior of interest}\}$$

For example, in order to detect a local concentration of $f(x)$ of the type similar to the crack tip stress field (Fig 6.12), one can define $t(x)$ as:

$$t(x) = \frac{a}{\sqrt{r}} \tag{6.17}$$

where a_0 is an intensity factor and r is the distance from the concentration point (in order to formally satisfy integrability criteria, $t(x)$ may be truncated at certain high levels). For the sake of clarity we assume here that the function t can be presented as a linear combination of certain primitive functions:

$$t(x) = \sum_{i=1}^{m} a_i t_i(x)$$

The next step is to select from the class T the function $\bar{t}(x)$ which is, in some sense, nearest to $f(x)$. If we define the error function:

$$e(x) = f(x) - t(x)$$

then the problem is to find $\bar{t}(x)$ which minimizes the L^2 norm of the error or, more conveniently, its square:

$$\|e\|^2 = \|f - t\|^2 = \int_{\Omega} [f(x) - t(x)]^2\, dx$$

Since the primitive functions $t_i(x)$ are predefined, this is equivalent to a simple finite-dimensional problem:

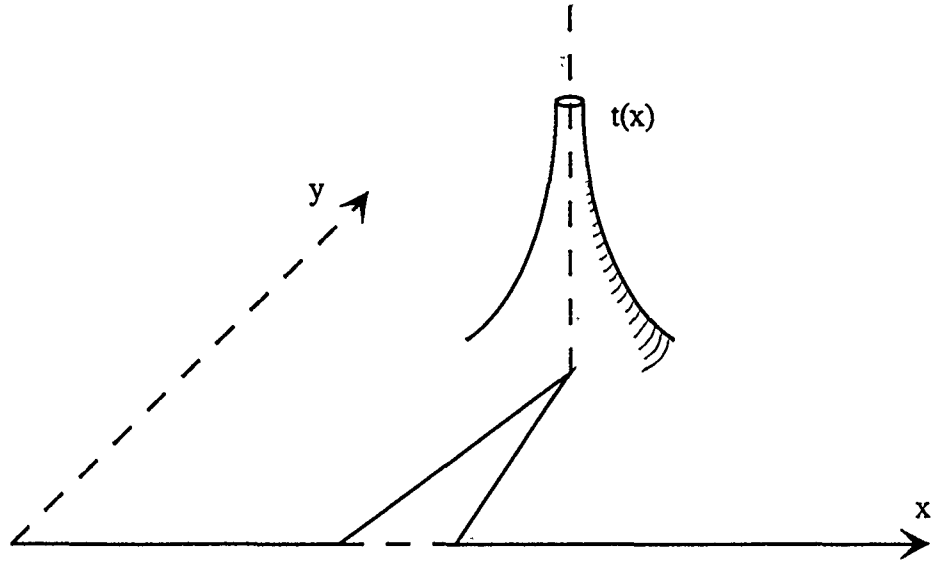


Figure 6.12: A pattern function for detection of stress concentrations.

Find $\overline{a} = (\overline{a}_1, \overline{a}_2, \ldots, \overline{a}_m)$ in $\mathbb{R}^m$ such that:

$$\|\overline{e}\|^2 = \min_{a \in \mathbb{R}^m} \|e\|^2$$

By standard arguments of variational analysis the solution to this problem is defined by the system of equations:

$$\frac{\partial \|e\|^2}{\partial a_i} = 0 \qquad i = 1, \ldots, m \tag{6.18}$$

which can be presented in the form:

$$A_{ij} a_j = b_i \qquad i = 1, \ldots, m$$

where

$$A_{ij} = \int_\Omega t_i(\boldsymbol{x}) t_j(\boldsymbol{x}) d\boldsymbol{x}$$

$$b_i = \int_\Omega f(\boldsymbol{x}) t_i(\boldsymbol{x}) d\boldsymbol{x}$$

The above integrals are well defined if $f(\boldsymbol{x})$ and all $t_i(\boldsymbol{x})$ are in $(L^2(\Omega))^N$, where N is the dimension of the euclidean space ($N = 1, 2$, or 3).

Once the function $t(\boldsymbol{x})$ is found, the proximity of $f(\boldsymbol{x})$ to the class T can be estimated by analyzing the relative error coefficient:

$$\varepsilon_t = \frac{\|e\|}{\|\overline{t}\|}$$

Note that this method can be extended to a more general form of $t(\boldsymbol{x})$, such as nonlinear combinations of primitive functions and coefficients a_i, differentiable with respect to a_i. For example, the class of singular solutions (6.17) can be made more general by introducing the arbitrary coordinates of the concentration point a_1, a_2, to give:

$$t(\boldsymbol{x}) = \frac{a_0}{\sqrt{(x_1 - a_1)^2 + (x_2 - a_2)^2}}$$

Then the system of equations (6.18) becomes nonlinear and can be solved by the Newton method. Note that the solution to (6.18) need not be unique in this case and may correspond to the local minimizer of the error norm.

In practical applications, the above analysis can be performed for several classes of test functions, each of them representing a certain specific behavior of f, such as pointwise stress concentrations, boundary layers, and other patterns.

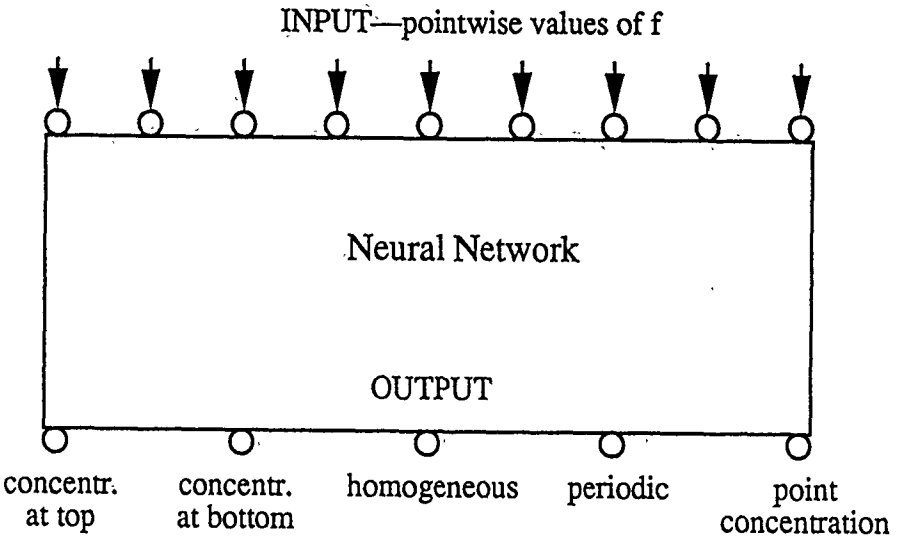


Figure 6.13: Application of neural network to recognition of patterns in the spatial distribution of function $f(x)$.

A qualitatively different method of detecting special features of the distribution of the function $f(x)$ in the domain Ω can be developed using the pattern matching capabilities of neural networks presented in Section 4. An idealization of such an application of neural networks is shown in Fig. 6.13.

The input layer of the network receives pointwise values of $f(x)$ in points distributed within the domain Ω. The consecutive neurons in the output layer correspond to the types of distribution which are of interest to the user. The network, after receiving the input values, fires the output neurode (or several neurodes) corresponding to the actual behavior exhibited by the function $f(x)$. The intensity of the output corresponds to the intensity of the correlation between $f(x)$ and the pattern represented by the output neuron.

6.3.2 Postprocessing in Solid Mechanics

In this section we will focus on postprocessing of finite element results in solid mechanics. We mean here the postprocessing for the purpose of application of knowledge engineering, namely the extraction of several values representing the most important characteristics of the solution. (Most of the formulas presented here were actually implemented in the expert system discussed in Section 8.)

Note that, in the spirit of hierarchical models discussed before, verification of the finite element results should be performed using the most comprehensive mathematical model available. This is because it is rather unreliable to qualitatively verify the results obtained with a simplified mathematical theory using the same mathematical theory. The verification will be more reliable if it is performed from the more general standpoint. For example, if the finite element results are obtained using the infinitesimal deformation theory, it makes sense to use large deformation theory for verifications. Note that there is no excessive computational cost involved, because the comprehensive theory is used for the postprocessing only, not for the solution of the problem.

Displacements

We are interested in the deformation of a material continuum, which at a certain initial time $t = 0$ occupies an initial configuration B_0, which is formally identified with the reference configuration B_R. The reference configuration is parametrized by the reference Cartesian coordinate system $\{X_K\}$ with base $I_K, K = 1, 2$ (see Fig. 6.14).

The reference coordinates X_I identify particles of a continuum in the sense that by particle X we mean the particle which in the reference configuration has a position defined by vector X.

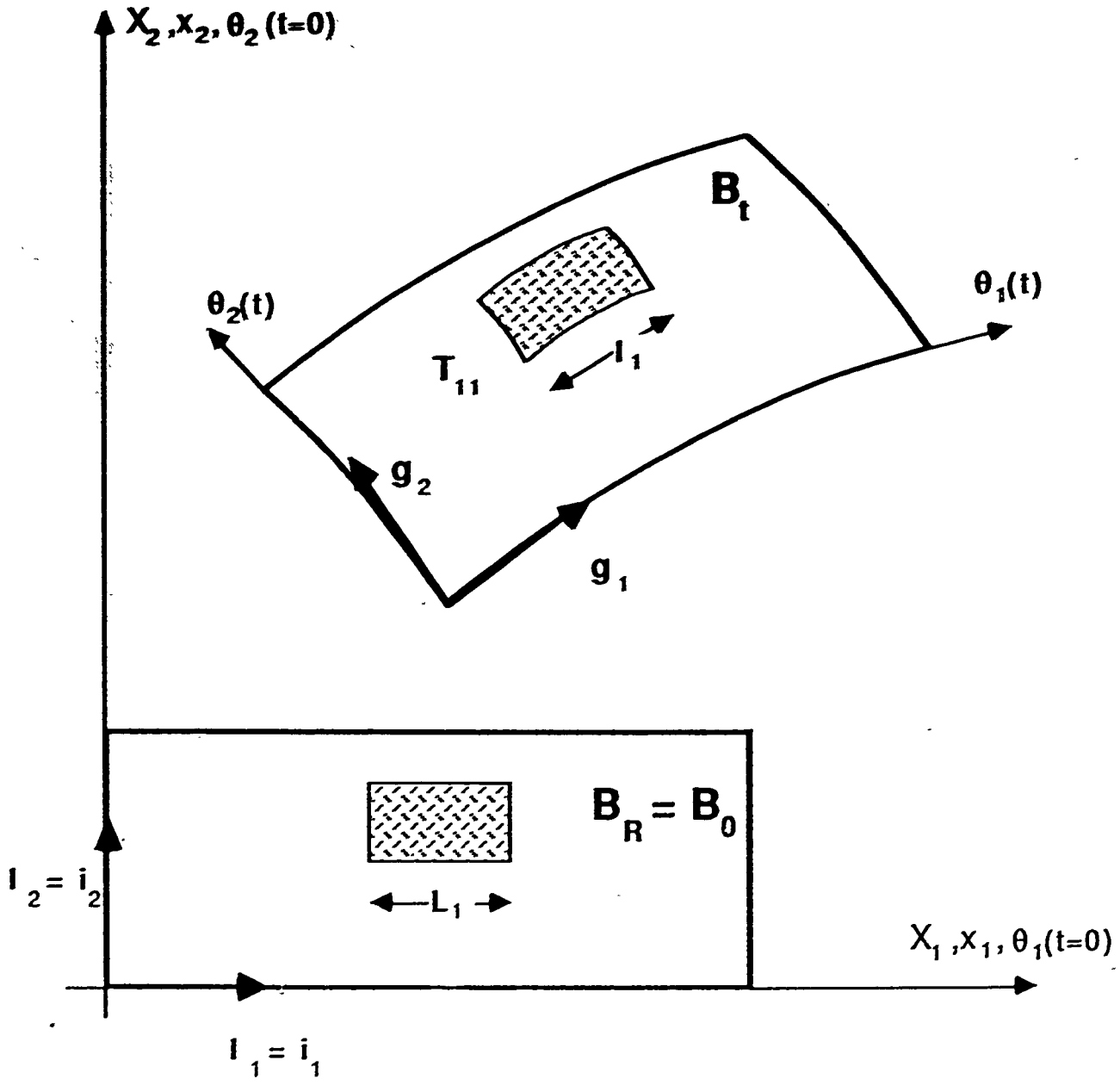


Figure 6.14: Deformation of a solid body.

At an arbitrary time $t > 0$ the continuum occupies a deformed configuration B_t, with new positions of particles referred to the Cartesian coordinate system $\{x_k\}$ with base $i_k, k = 1,2$. It is convenient, without restricting the generality of the description, to identify coordinate systems $\{x_k\}$ and $\{X_K\}$. The notion of systems $\{X_K\}$ and $\{x_k\}$ represents a typical notation convention of large deformation kinematics (see references [36,46]) in which *upper case symbols and indices refer to the reference configuration, while lower case symbols and indices refer to the current configuration.*

Motion of the continuum is described by the relation

$$x_i = x_i(X_I, t) \qquad t \geq 0$$

and the information about local deformation in the neighborhood of a particle X is presented by the deformation gradient tensor F with components:

$$F_{iJ} = x_{i,J}$$

where a comma denotes partial differentation.

Displacements of a material continuum are defined as

$$u(X) = x - X$$

For the verification of the results we are primarily interested in the magnitude of displacements, defined pointwise as

$$|u(X)| = \sqrt{u_i(X)u_i(X)}$$

For such a defined magnitude of displacements the usual minimum, maximum, and average values can be calculated using the formulas from the previous section. In the case of dynamic problems, similar calculations can be applied to velocity and acceleration fields.

Strains

In large deformation theory the components of the Green-Saint Venante strain tensor are defined as:

$$E_{IJ} = \frac{1}{2}(u_{i,J} + u_{j,I} + u_{k,I}u_{k,J}) \qquad i = i = J, j = J$$

If the deformation is infinitesimal, these components converge to the linearized strain measures:

$$\varepsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i})$$

Note, however, that if the deformations are large, the familiar interpretation of the strain measures as extensions of material fibers and changes of angles between fibers is not valid

anymore. To define physically meaningful measures of strain, the actual stretches of fibers need to be calculated:

$$\frac{l_I - L_I}{L_I} = \sqrt{E_{II} + 1} - 1 \tag{6.19}$$

In practice we are usually interested in maximum and minimum stretch in the body. These values can be calculated pointwise by application of formula (6.19) to the engenvalues of the strain tensor $\boldsymbol{E}$. These values can further be used to analyze the distribution of strains within the domain Ω, as described above.

Rotations

The rotations of a continuum are defined by the rotation tensor, obtained from the polar decomposition of the deformation gradient $\boldsymbol{F}$:

$$\boldsymbol{F} = \boldsymbol{R}\boldsymbol{U}$$

where $\boldsymbol{U}$ is a symmetric positive definite stretch tensor and $\boldsymbol{R}$ is the rotation tensor. The method of calculating the components of $\boldsymbol{R}$ from the components of $\boldsymbol{F}$ is straightforward and can be found in many references. See for example [36,46]. The finite rotation tensor $\boldsymbol{R}$ can be more intuitively represented by introduction of the rotation axis e_3 and the rotation angle ω:

$$\boldsymbol{R} = e_3 \otimes e_3 + \cos\omega\,(e_1 \otimes e_1 + e_2 \otimes e_2) - \sin\omega\,(e_1 \otimes e_2 - e_2 \otimes e_1)$$

Here e_3 is the eigenvector corresponding to the (only) real eigenvalue of $\boldsymbol{R}$ and e_1, e_2 are selected to form an orthogonal triad. The rotation about the axis e_3 is conveniently represented by the rotation vector:

$$\boldsymbol{\Omega} = \sin\omega\; e_3$$

The parameter ω can be directly interpreted as the rotation angle (in radians). In the case of plane problems the rotation axis e_3 is normal to the plane and ω represents the in–plane rotation. The distribution of the rotation angle $\omega(\boldsymbol{x})$ in the domain Ω can be analyzed using the generic methods introduced in Section 6.3.2.

In the case of infinitesimal deformations, the infinitesimal rotation vector is defined as:

$$\omega = \frac{1}{2}\,\mathrm{curl}\;\boldsymbol{u} \tag{6.20}$$

The magnitude ω of the rotation vector has the same interpretation here as in the case of large deformations. However, the formula (6.20) is valid only for small rotations (order of few degrees). Therefore it is sufficient only for verification of occurrence of larger rotations, but cannot be used to estimate the magnitude of rotations if they are actually large.

Rigid Body Rotations

One of the purposes of automatic verification of numerical results is to detect rigid body type modes of deformation, which may occur due to insufficient supports or insufficient connections between the structural elements. The displacement and rotation analysis described above can be used to detect the rigid body mode by checking the distribution of these parameters throughout the domain Ω. However, application of this approach to rotations has several disadvantages, namely:

- **it is rather computationally expensive, especially in the large deformation version, and**
- **the pointwise rotations are calculated as the combination of derivatives of the displacements. It is well known that the derivatives of finite element solutions are discontinuous across element boundaries, unless relatively expensive additional postprocessing is used. Therefore it may be difficult to filter the rigid body mode from the noise generated by the approximation error.**

That is why an alternative approach to extracting rigid body modes was considered, which is based on the postprocessing of displacements rather than their derivatives. This approach is based on representation of the rigid body motion in the form:

$$\widehat{\boldsymbol{u}}(\boldsymbol{x}) = \boldsymbol{u}_o + \boldsymbol{R}\boldsymbol{r}(\boldsymbol{x})$$

where $\boldsymbol{R}$ is a rotation tensor, $\boldsymbol{u}_o$ is a displacement of the reference point $\boldsymbol{o}$ (say the center of mass), and $\boldsymbol{r} = \boldsymbol{x} - \boldsymbol{o}$. For the actual displacement field $\boldsymbol{u}(\boldsymbol{x})$ one can define the L^2 error norm between the rigid body motion and the motion of the continuum as:

$$\|e\|^2 = B = \frac{1}{2}\int_\Omega [\boldsymbol{u}(\boldsymbol{x}) - (\boldsymbol{u}_0 + \boldsymbol{R}\boldsymbol{r}(\boldsymbol{x}))]^2 \, d\Omega \tag{6.21}$$

By the minimization of the norm one can find the rigid body translation and rotation. In fact, the displacement of the reference point $\boldsymbol{u}_0$ is immediately found as the pointwise value of the displacement. Thus, for plane problems, the only unknown in this problem is the rotation angle ω. This angle can be found from a standard minimization procedure

$$\frac{\partial B}{\partial \omega} = 0$$

or after substitution:

$$\int_\Omega \left\{ [\boldsymbol{u}(\boldsymbol{x}) - (\boldsymbol{u}_0 + \boldsymbol{R}\boldsymbol{r}(\boldsymbol{x}))] \frac{\partial \boldsymbol{R}}{\partial \omega} \boldsymbol{r}(\boldsymbol{x}) \right\} d\Omega = 0$$

This is a single nonlinear equation for ω, which can be solved by a Newton procedure:

$$\begin{aligned} k^n \Delta\omega &= 1^n \\ \omega^{n+1} &= \omega^n + \Delta\omega \end{aligned}$$

Here k is a single coefficient calculated as:

$$k = \int_\Omega \left\{ \left[\frac{\partial R}{\partial \omega} r \right] \cdot \left[\frac{\partial R}{\partial \omega} r \right] + [u(x) - (u_0 + Rr)] \cdot \frac{\partial^2 R}{\partial \omega^2} r \right\} d\Omega$$

and 1 is a residual:

$$1 = - \int_\Omega \left\{ [u(x) - (u_0 + Rr)] \cdot \frac{\partial R}{\partial \omega} r \right\} d\Omega$$

The integrals in the above formulas can be calculated using standard finite element integration. The matrices R, $\frac{\partial R}{\partial \omega}$, and $\frac{\partial^2 R}{\partial \omega^2}$ are defined for two-dimensional problems as:

$$R = \begin{bmatrix} \cos\omega & -\sin\omega \\ \sin\omega & \cos\omega \end{bmatrix}$$

$$\frac{\partial R}{\partial \omega} = \begin{bmatrix} -\sin\omega & -\cos\omega \\ \cos\omega & -\sin\omega \end{bmatrix}$$

$$\frac{\partial^2 R}{\partial \omega^2} = -R$$

The departure of the actual displacement field from the rigid body mode can be estimated by the calculation of the error norm in (6.21). In practical applications the extraction of the rigid body mode should be performed for structural elements (subregions of Ω) rather than for the whole domain.

Stresses

In the verification of stresses we are primarily interested in the actual stress state on a deformed body, represented by the Cauchy stress tensor:

$$T = t_{ij} e_i \otimes e_j$$

In many finite element computations, especially in large deformation elasticity, a second Piola-Kirchhoff stress tensor is used, defined as:

$$\hat{T} = t_{IJ} e_I \otimes e_J$$

The components of this tensor, although convenient for the solution of the problem, have basically no direct physical interpretation. Therefore for verification purposes they must be transformed to the Cauchy form, using the formula:

$$\boldsymbol{T} = \frac{1}{\det \boldsymbol{F}} \boldsymbol{F} \hat{\boldsymbol{T}} \boldsymbol{F}^T$$

Since the full stress tensor in general has six independent components, it is convenient to introduce a single measure of stress intensity level, such as the stress intensity factor, defined as:

$$\sigma_i = \sqrt{3 J_2}$$

where J_2 is the second invariant of the stress deviator. Equivalently, principal stresses or other parameters can be used for verification. The distribution of the stress intensity throughout the domain Ω can be analyzed using methods found in Section 6.3.2 and then furnished to the verification expert system for checking and evaluation.

6.4 Automatic Verification of Numerical Results

In an automated environment for engineering design the results obtained with the finite element method or other numerical techniques should be automatically verified for consistency with the mathematical formulation (in particular assumptions), satisfaction of various design criteria, errors introduced by the discretization, etc. Here we assume that the approximation error has been taken care of by the adaptive mesh refinement procedure, so that the error associated with numerical modeling is below certain prescribed threshold value. Still, the solution may be incorrect due to:

- incorrect selection of the mathematical model,
- incorrect specification of boundary conditions, such as insufficient support, excessive loads, etc.,
- incorrect selection of material properties, and
- other modeling errors.

The information obtained from generalized postprocessing can be used to automatically verify the finite element results and possibly automatically modify the model in case there is a violation of certain verification criteria. The automated verification of the results is performed by their comparison with the assumptions of the mathematical model, basic design criteria, or existing database related to the problem being solved (we will not discuss here

the detailed verification criteria). The expert system implemented for this purpose in this project is presented in detail in Section 8.

The results of automated verification of the numerical solution can be used as the basis for automated modification of the structure or the mathematical model in order to satisfy the design criteria. In general, the decisions involved here will be very complex and strongly dependent on the characteristics of the class of problems to be solved. As an example, consider the situations when the constitutive model used in the analysis is linear and the value of stress calculated exceeds the plastic yield limit. In such a situation there are at least three possible solutions:

1. If plastic deformations are acceptable in the structure being designed, change the constitutive equations to elasto-plastic and repeat the analysis.
2. If plastic deformations are not acceptable, but there exists a variety of materials from which to choose, switch to the material with better mechanical properties.
3. If there is no possibility of selecting better material, try to redefine the shape of the model.

Clearly the selection of a particular solution depends strongly on the specific design considered. In specialized applications the expert system should be able to automatically make appropriate selections. However, in more general applications it will be very difficult to reasonably choose the correct solution. That is why in the verification system implemented in this project the possible reasons of violation of the design criteria are presented to the user together with possible (and available) methods for fixing the problem. However, it is up to the user to select the actual approach. The details of this procedure are discussed in Section 8.

7 A General Computational Environment for Automated Structural Analysis

Development of a computational environment for automated structural design is a complex task, requiring a combination and integration of various—functionally very different—pieces of software. At the present state of the software "market" this includes:

- CAD and solid modeling programs
- Finite element mesh generators
- Finite element analysis programs

- **Post-processors**
- **Knowledge-based expert systems**
- **Other elements (spreadsheets, knowledge acquisition, neural networks, etc.)**

The first step in the design of the automated design environment is the general layout of basic components, taking into account both their functional characteristics and the data types accessed by each component. In the following two sections a general structure of the proposed environment will be discussed.

7.1 Computational Environment—Functional Structure

The basic role of the components of the automated design environment is to aid or replace the human designer at consecutive stages of the design process, as presented in Section 3, Fig. 3.1. Therefore, the general functional structure of this environment is somewhat similar to the flow chart of the design process. This general structure is presented in Fig. 7.1, where the consecutive blocks indicate separate elements (pieces of software or their combinations) that are supposed to aid or replace engineers in their work.

The first stage of the analysis, namely the general comprehension of the physical problem and the objectives of the design, is generally performed by the designer (although some help from an intelligent system may be used at this stage). The next step is the construction of the general structural model, including identification of various structural elements, loads, supports, etc. At this stage, the assistance of CAD programs and an intelligent advisor (KBES) is recommended. It is of importance to note that this advisor should definitely possess elements of the object oriented approach (OOP) to provide a natural classification and identification of elements of the structure. The third step of the design is the construction of the mathematical model of the problem. The structure will be viewed as a certain domain with prescribed equations, boundary conditions (both supports and applied tractions) and interface conditions for various structural elements (subdomains). At this stage, the level of mathematical complexity of the model will be decided, for example, the use of large deformation or small deformation theory, application of beam or solid body formulation, etc. The aid to the designer at this stage should be provided by an expert system (KBES) with object-oriented capabilities. Note that the knowledge basis of this system will include—in the general case—both heuristic knowledge and more precise estimates based on appropriate theories discussed in Section 6.1. Therefore this system should be capable of performing basic algebraic operations.

In the general situation a simplified analytical analysis of the introductory design may be performed at the first pass through the design process, in order to avoid an expensive finite

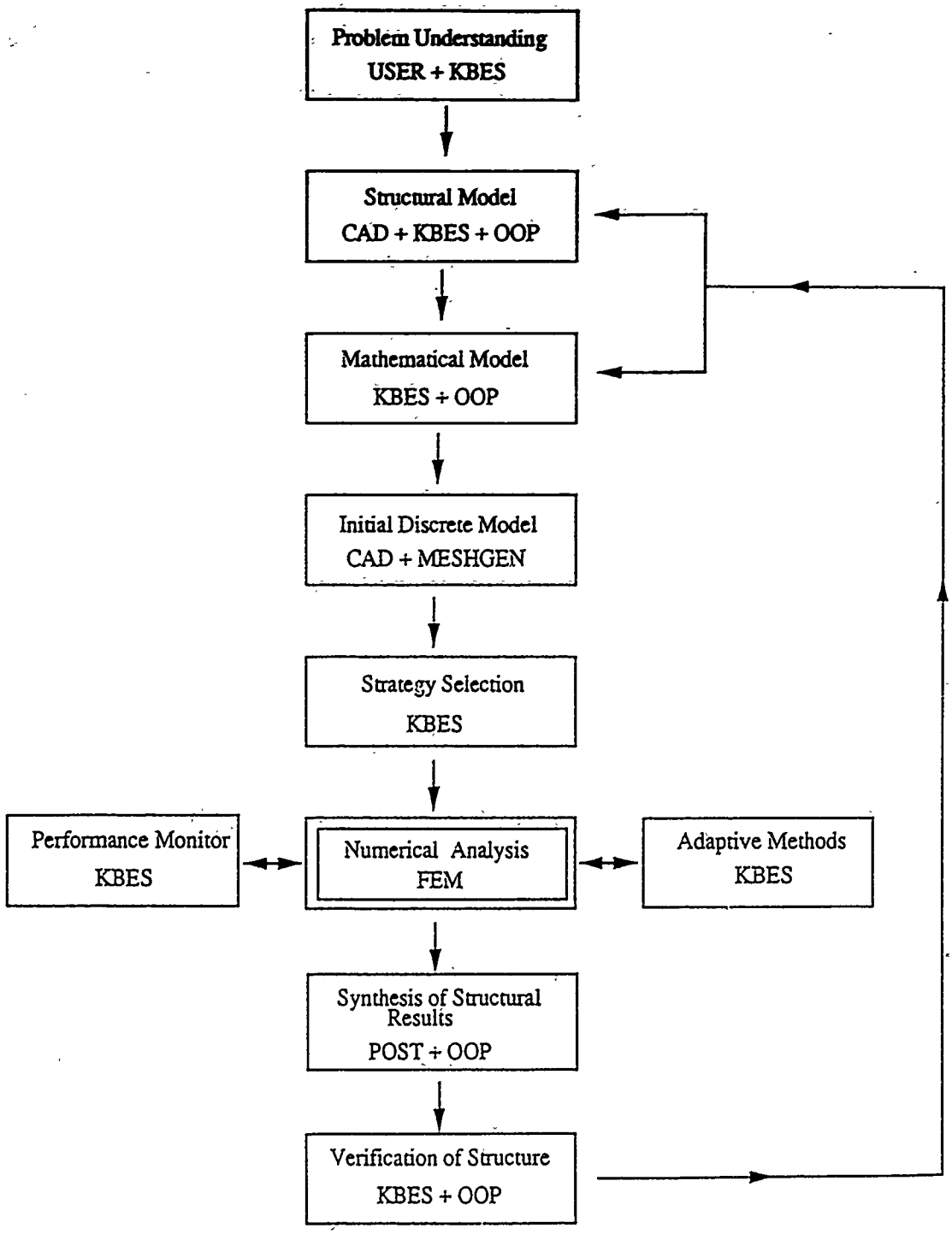


Figure 7.1: The general automated computational environment—functional structures

element analysis of models that are far off the appropriate design. Here we will assume for simplicity that after the construction of the structural model the full finite element analysis is performed. Therefore the next stage of the design includes generation of the discrete mesh for the structure (or, more formally, for the domain of the mathematical model). At this stage a combination of CAD software with mesh generation programs can be used. It is also possible to use a small advisory intelligent system or intelligent CAD environment [57].

It should be noted that, if the finite element software is equipped with adaptive capabilities (as is the case in this project), the role of the initial mesh is merely to provide an adequate representation of the shape of the domain (the mesh will be adjusted to the characteristics of the solution during the solution process). Without these adaptive capabilities the initial mesh generation is an extremely difficult task, because the characteristics of the final solution must be guessed *a priori* to provide mesh concentration in regions of high gradients of the solution and other critical zones.

After the generation of the initial mesh, the solution of the problem by the finite element method is performed. However, most of today's large finite element codes offer a selection of strategies for the solution of the problem—for example various methods of time integration, eigenvalue extraction, or the solution of the linear system of equations. Usually the selection of the method best suited for the particular example depends on the characteristics of the problem, for example the number of nodes, types of loads, etc. In complex, nonlinear problems, selection of the proper strategy is often crucial for the achievement of the final solution to the problem. In general, a considerable expertise and experience is required at this stage and the application of a competent expert system is very desirable. It is not necessary for this system to have object-oriented capabilities but, on the other hand, learning and knowledge acquisition capabilities will be very useful.

The central element of the analysis process is the solution of the discrete model by the finite element method. Usually, finite element codes are viewed as "black boxes" with virtually no decisions to be made after the initiation of the program. However, this is not necessarily true, especially if the adaptive or "smart" methods are being used and if the problems solved are characterized by strong nonlinearities. Some of the examples of problems that require decision making (including heuristic decisions) during the finite element analysis include:

- Solution of strongly nonlinear or time-dependent problems.
- Application of iterative procedures (e.g., for the solution of linear systems of equations).
- Adaptive mesh refinement.
- Adaptive zonal methods (e.g., implicit/explicit algorithms)

On the chart of Fig. 6.1 the decision–making modules of the finite element analysis are represented by two Knowledge–Based Expert Systems:

1. A performance monitor KBES is responsible for monitoring the behavior of various iterative procedures (e.g., Newton method for nonlinear problems, Jacobi conjugate gradient method for linear system of equations, etc.) and taking appropriate actions if any computational problems occur.

2. A discrete model modification monitor KBES is responsible for providing additional intelligent support for adaptive mesh refinement, zonal methods, and other methods oriented on achieving high quality results. It should be noted that these methods are usually quite precise and algorithmic in nature, yet some heuristic knowledge is usually necessary to handle nontrivial cases and improve the overall performance.

In more general cases other advisory expert systems may be used. Usually it is not necessary for these systems to have object–oriented capabilities but, similarly as in the case of strategy selection, learning and knowledge acquisition capabilities are very desirable.

Once a high–quality finite element solution has been obtained, the results of the analysis have to be evaluated in the context of compliance with structural design criteria, like maximum deflection, maximum stress, safety coefficients, etc. This task can be performed by the structural evaluation KBES with object–oriented capabilities. Note that prior to application of such a system the essential information about the solution must be extracted from the massive finite element data. This task is performed by the specialized post–processing module (with strongly recommended object–oriented capabilities for natural representation of structural objects). Depending on the decision reached at this stage of structural evaluation, the model may be accepted or rejected. If the model or the solution is rejected, usually some modifications to the structural or mathematical model are recommended and implemented, and the next loop of design is performed.

The chart discussed here presents only the general types of software used in the automated computational environment and the flow of control between these modules. For practical applications the generic structure of the data (knowledge) needs to be formulated and the protocol of the exchange of information between different modules needs to be established. These issues will be discussed in the next section.

7.2 Computational Environment—A General Data Structure

The amount of data and information processed in the engineering design process is enormous. The data structure devised to handle this information should satisfy two basic criteria of somewhat contradictory nature, namely:

- easy access of every functional component to every essential piece of information, and
- encapsulation of knowledge, reducing to the necessary minimum the length of the information search pattern.

In order to facilitate satisfaction of these criteria it is of importance to observe that there are three basic groups of data (static knowledge) that are of different types and that can be effectively separated from one another. These are:

1. Information about the structure analyzed, models, loads, structural response, etc.
2. The finite element data concerning nodes, elements, integration points, values of the solution in these points, etc.
3. Information about performance of various strategies, e.g., histories of error tolerances in nonlinear problems, counters of time steps, measures of convergence of iterative processes, etc.

It is essential for the effective operation of the data structure that neither the designer nor the expert systems dealing with structural design should interact with the massive finite element data. The essential information about the structural elements (like maximum stress) should be derived from the finite element information by the post–processing module and only then evaluated by the designer or the specialized expert system. According to this remark, the general structure of the data should be of the type presented in Fig. 7.2. The figure also shows the consecutive functional components of the system (defined in Fig. 7.1) and the information accessed by these components.

After the general division of data into groups, the question arises: How should the data (static knowledge) be organized in each of these groups? The possible answers to these questions differ depending on the actual type of information. For finite element data a traditional way of storing the information is in the form of vectors and arrays. Depending on the specific application it can actually be implemented in the form of full vectors reserved in the memory, pointer lists, linked lists, or tree structures. Another option is based on the concepts of object–oriented programming.

As for the structural information, the question of the optimal data structure is more difficult due to a considerable diversity of structural forms, interfaces, loads, etc. Apparently the two options that are generic and flexible enough to handle this information are:

- The blackboarding technique, and
- The object–based data structure.

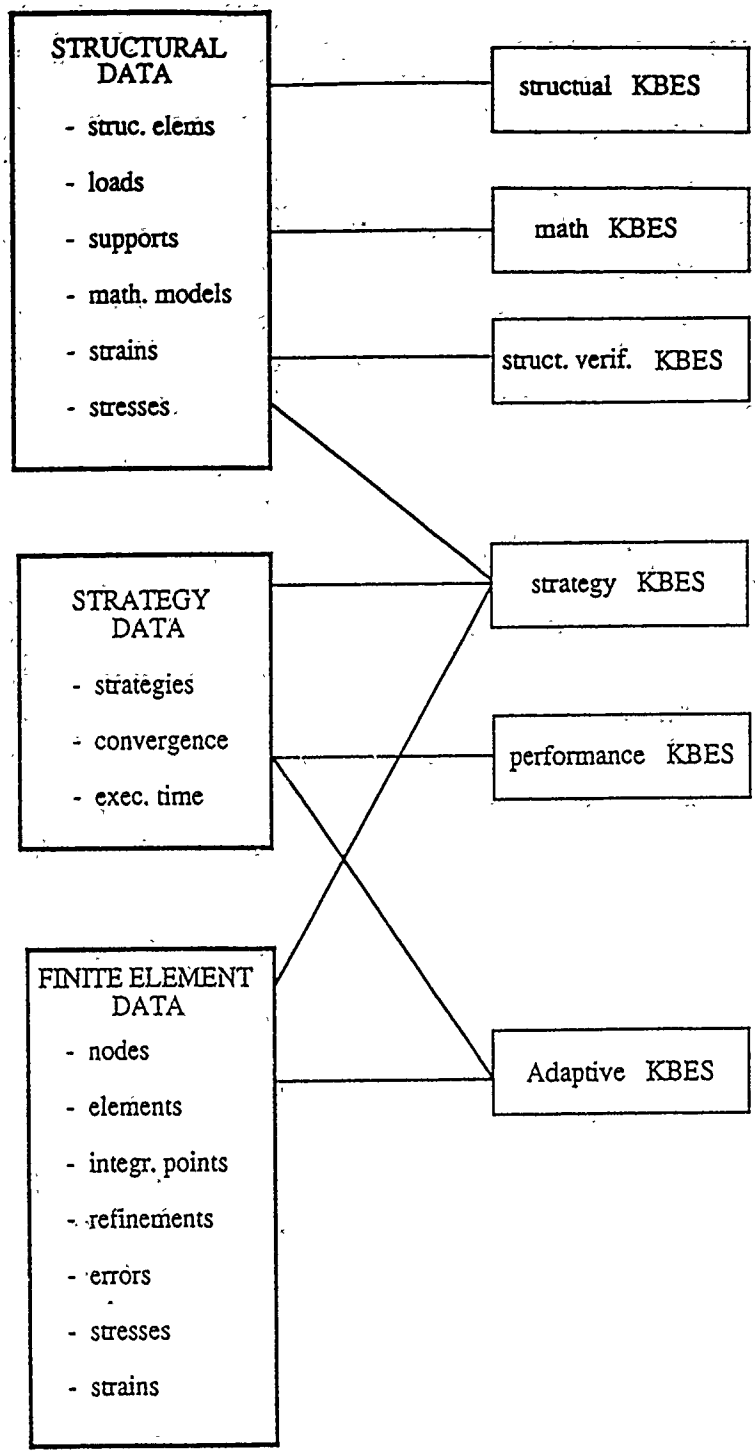


Figure 7.2: General types of data in the design based on the finite element method.

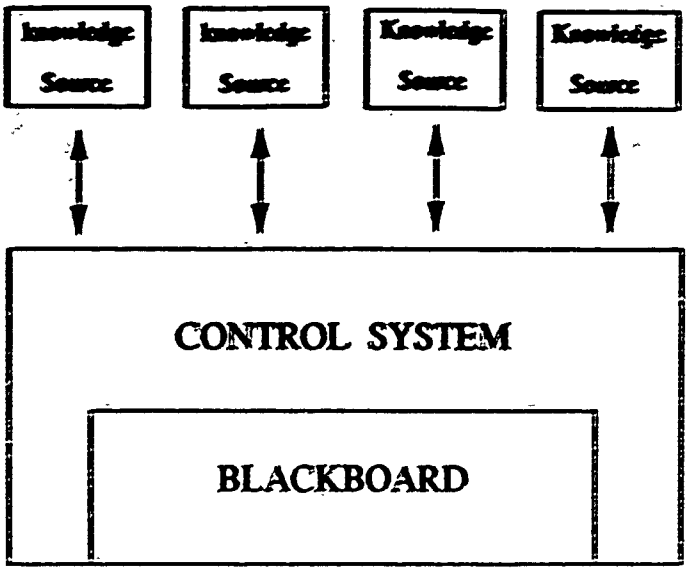


Figure 7.3: Typical structure of a blackboarding data storage.

The blackboarding technique [37,75] is based on the simple concept that all the pieces of information are written to the "blackboard" (which is actually a data storage system) and are available upon request to all the members of the session called knowledge sources (pieces of software in our case). The participating knowledge sources (see Fig. 7.3) may be very diverse, but they use the same protocol in the communication with the blackboard. Thus the information stored in the blackboard is available to all the sources and users of knowledge. The basic problem in the blackboard approach is the management of the blackboard itself if large amounts of information are processed.

Another attractive concept of the structure of static knowledge (and, for that matter, also of active knowledge methods) is the object based approach discussed in Section 5. This approach is based on the abstract data theory, in which it is not essential how the information is actually stored; the essential fact is who (what object) is responsible for providing this information. The information is actually requested by objects from other objects without concern about the actual type of storage used. Within this system the information is encapsulated within defined classes of objects, and at the same time the whole body of information is available to all objects. Due to numerous conceptual and practical advantages of this kind of knowledge structure, most of the recently developed advanced expert system shells (reviewed in Section 5) usually provide object-oriented capabilities. In this work, the object-oriented approach was used. A more detailed discussion and design of object-oriented approaches to knowledge engineering in automated computational mechanics will be discussed in the next section.

8 Design, Implementation and Examples of a Coupled FEM–KE Environment

8.1 Introduction

To show the practical feasibility of the concepts developed in this project, a fully coupled finite element–knowledge engineering environment was designed and implemented on a research scale. This implementation combines adaptive methods, smart algorithms, expert systems, and other engineering tools to provide maximum reliability, robustness and ease of use of the finite element software. Note that this effort is beyond the original statement of work.

The implementation presented here is based on the *h-p* adaptive finite element code PHLEX and the expert system shell NEXPERT OBJECT. It is important to note here that the previous report only presented an introductory general design of the environment considered in this section. This design was based on general concepts of object oriented programming, data management, and knowledge engineering. In the practical implementation discussed here some of these concepts have been changed, primarily due to functional limitations of the expert system software used in this project (not all generic concepts were available in this particular implementation). However, the general idea and practical functionality have not changed.

8.2 An Automated PHLEX–NEXPERT OBJECT Computational Environment

The automated environment for finite element analysis is based on full coupling of two different computer tools: an algorithmic finite element code PHLEX and an expert system shell NEXPERT OBJECT.

The PHLEX code developed at COMCO represents a new generation of the finite element software, based on *h-p* adaptive methods, error estimators and refinement strategies. The program is designed to achieve maximum reliability and quality of results at a minimum computational cost. This is achieved by implementation of rigorous error estimates, adaptive refinement strategies and smart algorithms.

NEXPERT OBJECT is a generic expert system shell developed by Neuron Data, Inc. and designed for customized construction of powerful expert systems for generic applications. The shell is equipped with object oriented capabilities (data representation) and a versatile inference engine.

The coupled finite element–knowledge engineering environment is based on direct inter-

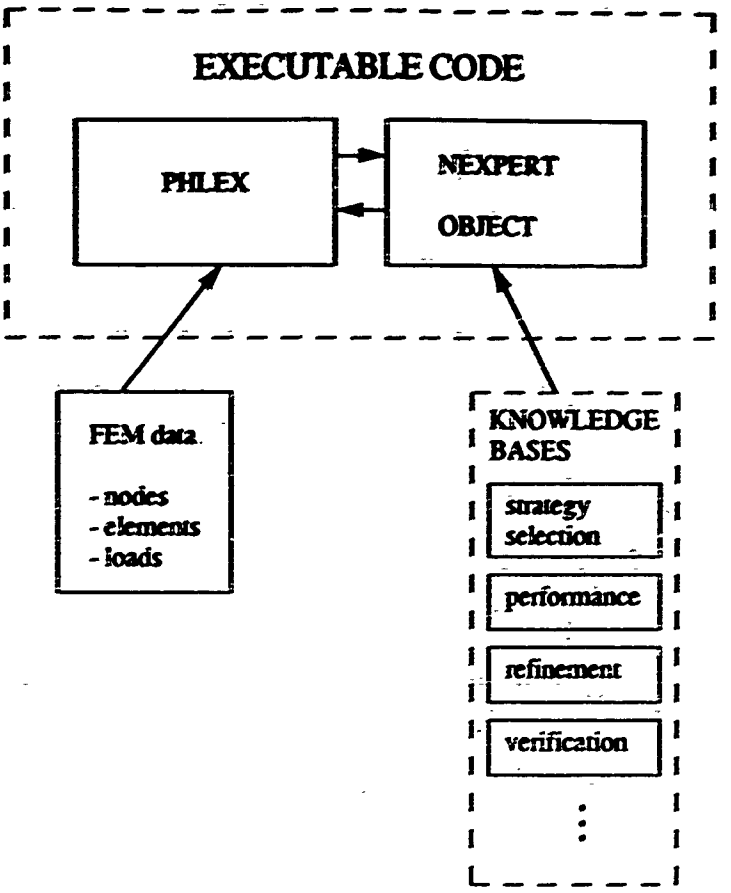


Figure 8.1: A general scheme of the coupled PHLEX–NEXPERT OBJECT environment.

action of the finite element program with the expert system. Within this environment, the algorithmic code can request advice from the expert system or, conversely, the inference engine can activate algorithmic procedures. The general scheme of this coupled environment is presented in Fig. 8.1.

The actual executable code includes both finite element procedures and knowledge engineering software. These two programs communicate through a well-defined interface. Most of the finite element software was written in FORTRAN, and the expert system shell was written in C. Therefore the actual interface includes an additional translator designed to handle language differences. The data for the finite element code (mesh data, loads, etc.) is stored in separate data files or is provided by the user during execution. The actual expert knowledge, represented in the form of classes, objects and rules, is stored in separate knowledge bases. This enables easy substitution of the expert information for different applications of the software.

Both the finite element software and the expert system shell operate in the interactive

graphic mode based on a windowing system. A sample copy of the computer screen during the development of knowledge bases is presented in Fig. 8.2. The figure shows both the PHLEX window (large, central position) and several windows of the expert system shell. Within this environment it is possible to develop and test the knowledge bases when the expert system is called by the finite element program. Such a situation is presented in Fig. 8.3, which shows an additional rule network window with graphical presentation of rules and the current status of the inference engine. This graphical interface facilitates the development of knowledge bases.

Once the development of knowledge bases has been completed, the expert system graphical interface is deactivated and the expert system runs in the background. This is the version that can be delivered to the user of the system.

Within this generic environment, several expert systems were implemented to assist the finite element program in automated performance of consecutive stages of the analysis. A more detailed presentation of implementation and performance of some of these systems is discussed in the following sections.

8.3 Automated Strategy Selection and Performance Monitoring

Advanced finite element programs are usually equipped with a variety of computational strategies for the solution of the problem. The performance of these methods depends upon various parameters of the problem under consideration. Therefore selection of the most appropriate method is a complex and rather heuristic task, very difficult for an inexperienced user. During the execution, many computational methods need to be closely monitored and have certain parameters adjusted in order to assure stability, convergence, and good performance. A typical example is the solution of strongly nonlinear problems by Newton–type methods.

In this section we present a research–scale implementation of expert systems devised to automatically select computational strategies and to monitor the performance of these strategies during the solution process. This implementation is based on the concepts discussed in Section 6.

8.3.1 Selection of Computational Methods

An expert system was implemented to automatically select the type of solver for the linear system of equations resulting from the finite element discretization. The system selects automatically either the frontal or the iterative solver. This selection is based on simple criteria, such as the number of degrees of freedom and the ordering of the elements. The

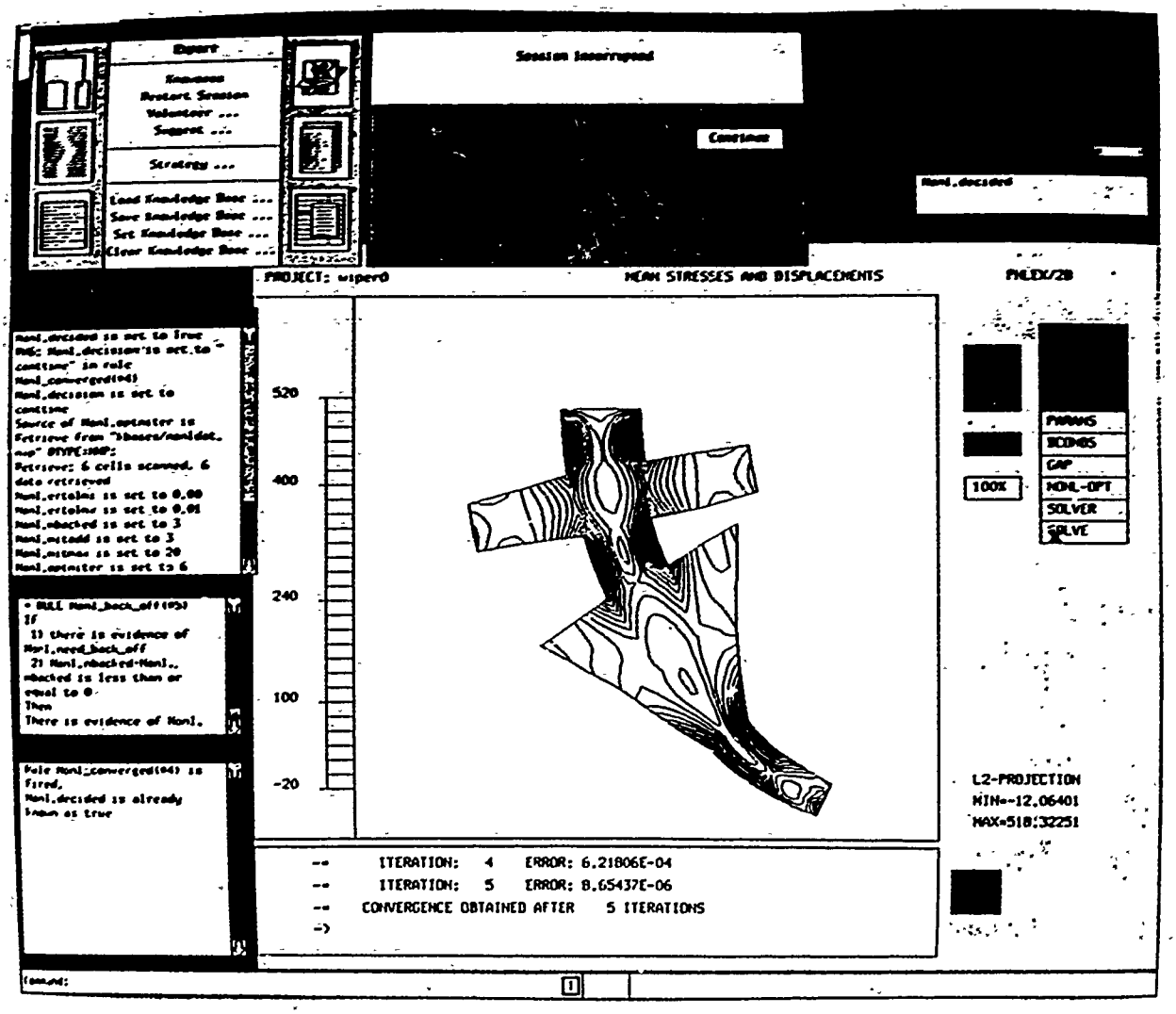


Figure 8.2: Computer screen with PHLEX and NEXPERT windows.

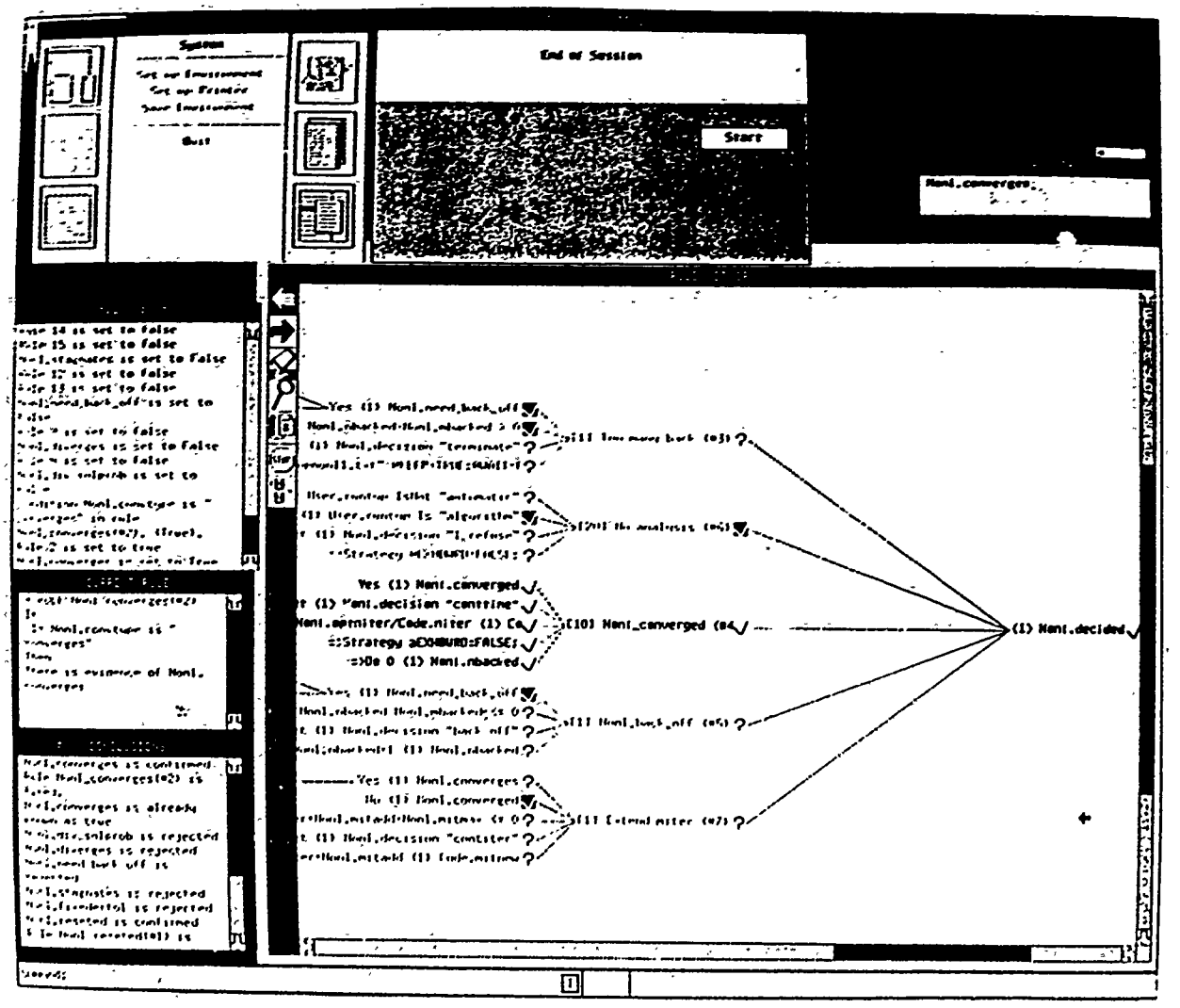


Figure 8.3: Testing knowledge bases in the interactive mode.

listing of objects and rules of this expert system is presented in Appendix C1.

The automated selection of the solver is activated by the user's command:

SOLVER,AUTOMATIC

issued to the finite element program.

8.3.2 Performance Monitoring and Control

A performance monitor expert system is designed to monitor the performance of various sections of the finite element code, inform the user about occurring abnormalities (divergence, zero pivots, slow convergence, etc.) and to suggest or perform appropriate measures to fix the problem. The importance of such an intelligent advisor is especially pronounced in large finite element programs for nonlinear or time–dependent problems, with a multitude of methods available for the solution of linear systems of equations, eigenvalue extraction, etc.

The general structure of an automated performance monitor is presented in Fig. 8.4. This is an advisory system to the finite element code which is used for the solution of nonlinear or time–dependent problems, with a selection of either a direct (frontal) solver or an iterative solver for the linear system of equations. The structure here utilizes the concepts of object oriented programming to represent specialized knowledge of several experts. The actual concept presented in Fig. 8.4 represents, in an object oriented fashion, the following situation:

> Image a finite element code running on the computer and printing to the screen an echo of the performance of various methods used in the code (execution times, numbers of iterations, error tolerances, etc.). In front of this screen there sits a number of "experts", each of them specializing in a different method. In particular there is a specialist on nonlinear problems, a specialist on iterative solvers, and others. Each expert analyzes the information pertaining to his discipline and derives conclusions concerning the performance of the code (and methods of fixing possible problems). The information and suggestions provided by each expert are analyzed by the supervisor (general performance monitor) which makes a final decision concerning continuation of the computations, change of strategy, or termination of the computations in "hopeless" cases.

The classes presented in Fig 8.4 represent formally the human experts. Note that intermediate classes are introduced in this scheme to capture methods common to more than one

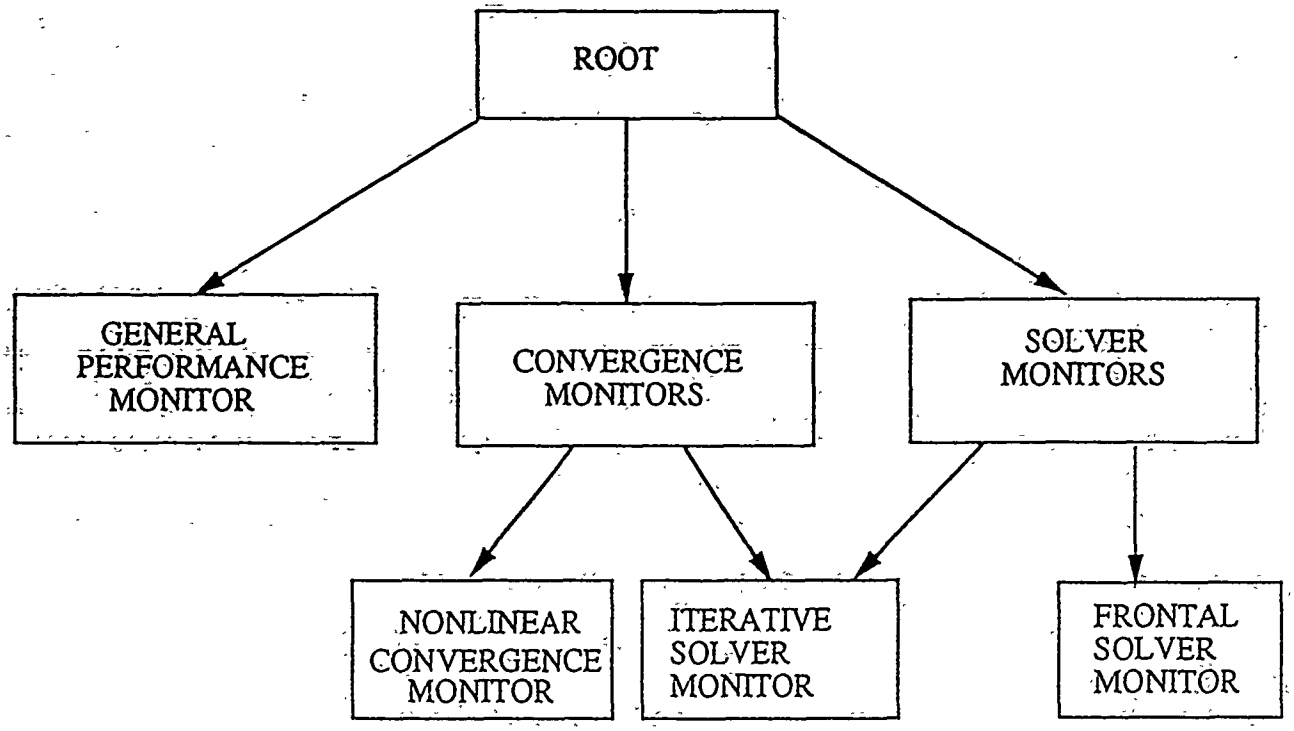


Figure 8.4: A general structure of the Performance Monitor expert system.

expert. For example, the method of convergence checking for nonlinear problems is very similar to the convergence check used for iterative solvers. Therefore this common method is represented by a general "convergence monitors" class.

In this project only one expert system was actually implemented and tested, namely the nonlinear monitor expert system. The principles of operation are discussed in Section 6. The listing of objects and rules of this system is presented in Appendix B2. The expert system is activated at each load step after completing a prescribed number of iterations (sufficient to estimate trends in error histories). The decisions of the expert system are used to control the solution process and obtain a converged solution at minimum cost.

The automated performance monitoring option is activated by simply specifying AUTOMATIC on the list of commands for the finite element nonlinear analysis.

8.3.3 Examples

To illustrate the practical performance of the strategy selection and performance monitoring expert systems we present a numerical analysis of a windshield wiper blade in contact with a rigid surface. The deformation of the blade is described by a large deformation theory. This theory, combined with unilateral contact conditions, leads to a strongly nonlinear problem.

This problem was solved on a mesh with both h and p refinements (Fig. 8.5) and a nonlinear Newton procedure with incremental loading was used. The increasing load was controlled by a quasistatic time and corresponded to pressing the blade against the windshield.

A copy of the echo of the solution process is presented in Fig. 8.6. The frontal solver was automatically selected for this relatively small problem. Then, at the first load step, the Newton procedure did not converge within the prescribed number of iterations. Still, the error history suggested that the process was actuallv convergent, therefore the expert system decided to perform a few additional iterations in o.der to obtain full convergence. However, these additional iterations did not fully succeed. After this situation had occurred several times, the expert system concluded that it didn't make sense to continue iterations. The computations were backed to the last converged solution (initial guess in this case) and repeated with a smaller time increment. After a few further adjustments, a sequence of convergent iterations was reached and the final solution was obtained (only a few of the initial iterations and the final iteration are shown in Fig. 8.6). The final deformed configuration with stress intensity contours is presented in Fig. 8.7. Note that if a user-prescribed fixed load step was used, a large number of small steps was required to obtain convergence or the process diverged.

The above example presents the most typical result of the application of the perfor-

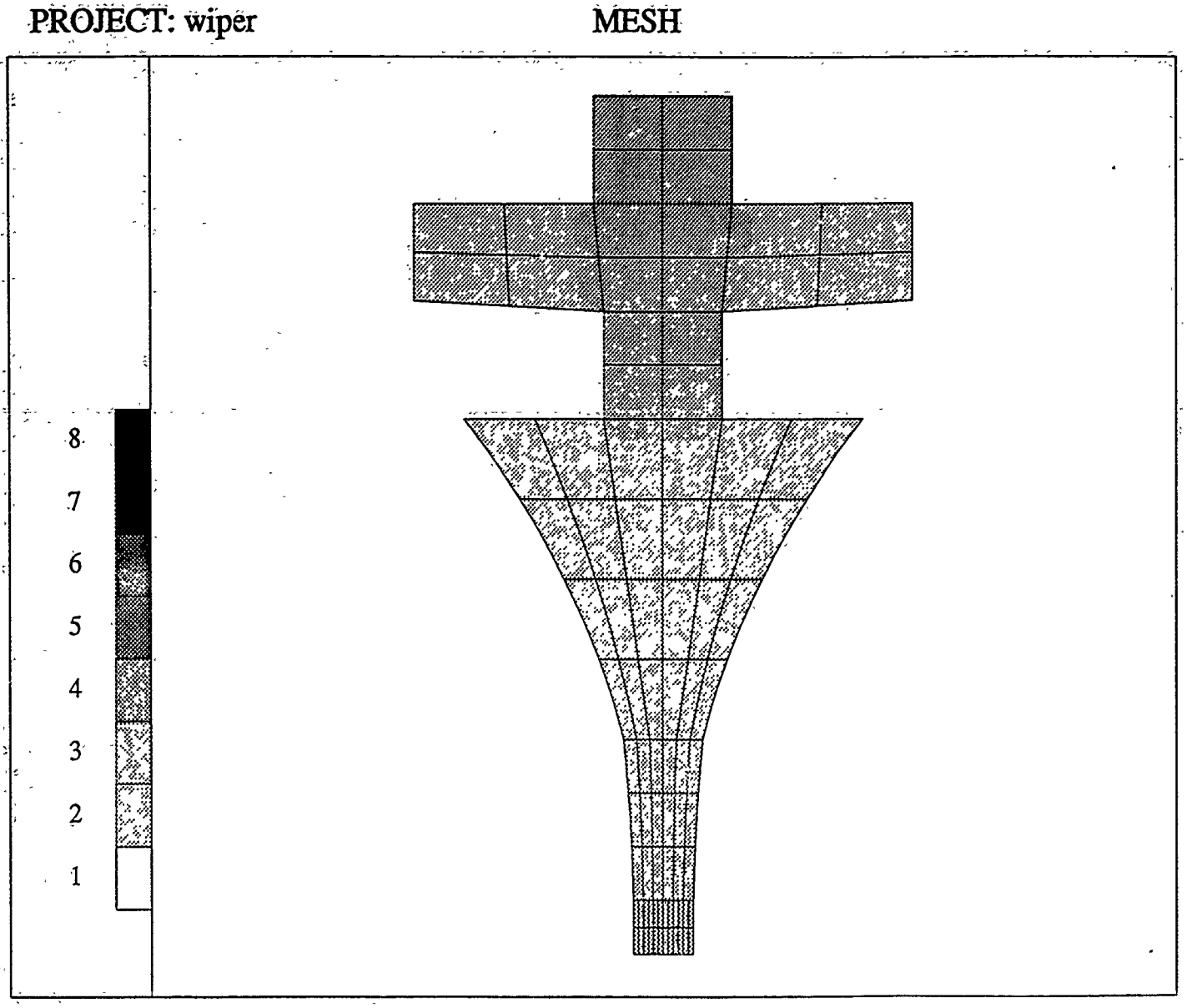
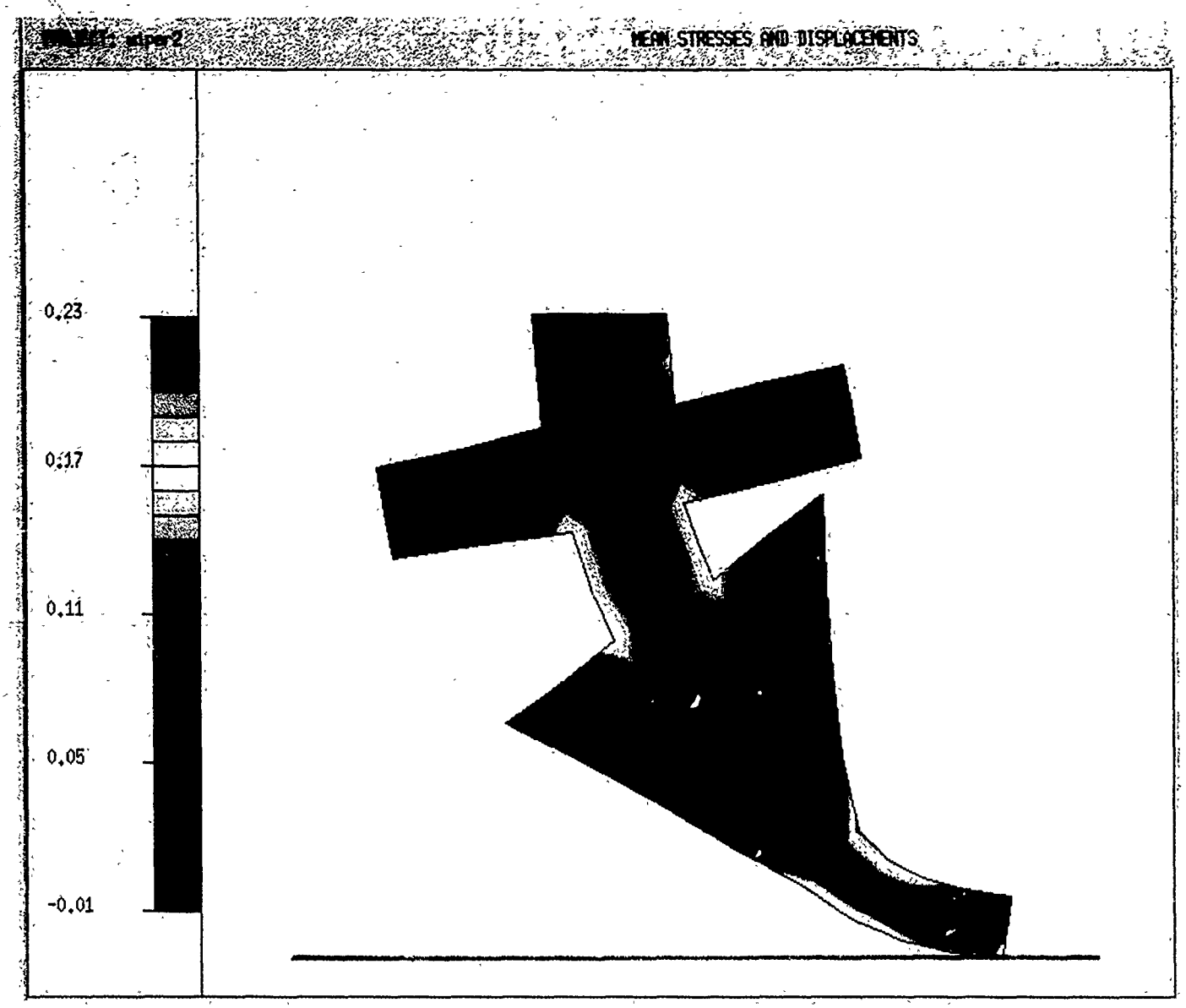


Figure 8.5: An *h-p* finite element mesh for the wiper blade analysis. Shade intensity represents the order of approximation.

```
EXPERT: SOLVER SELECTED = FRONTAL

NONLINEAR PROCESS:

   CURRENT TIME  =  5.0000000E-01     DT =  5.0000000E-01
     ITERATION:   1    ERROR: 1.00000E+00
     ITERATION:   2    ERROR: 6.85089E-02
     ITERATION:   3    ERROR: 9.00390E-02
     ITERATION:   4    ERROR: 4.20841E-02
     ITERATION:   5    ERROR: 1.70250E-01
     ITERATION:   6    ERROR: 8.15674E-03
     ITERATION:   7    ERROR: 5.84537E-02
     ITERATION:   8    ERROR: 2.93982E-02
     ITERATION:   9    ERROR: 1.54566E-02
     ITERATION:  10    ERROR: 6.95750E-03
   WARNING: NO CONVERGENCE AFTER   10 ITERATIONS
   EXPERT DECISION: CONTINUE ITERATIONS
     ITERATION:  11    ERROR: 4.05248E-03
     ITERATION:  12    ERROR: 8.20585E-04
     ITERATION:  13    ERROR: 4.98021E-03
   WARNING: NO CONVERGENCE AFTER   13 ITERATIONS
   EXPERT DECISION: CONTINUE ITERATIONS
     ITERATION:  14    ERROR: 3.61173E-03
     ITERATION:  15    ERROR: 2.29691E-03
     ITERATION:  16    ERROR: 5.19300E-03
   WARNING: NO CONVERGENCE AFTER   16 ITERATIONS
   EXPERT DECISION: BACK OFF, ADJUST DT AND RETRY

   CURRENT TIME  =  2.5000000E-01     DT =  2.5000000E-01
     ITERATION:   1    ERROR: 1.00000E+00
     ITERATION:   2    ERROR: 3.39861E-02
     ITERATION:   3    ERROR: 5.77968E-02
     ITERATION:   4    ERROR: 2.28434E-02
     ITERATION:   5    ERROR: 3.76144E-02
     ITERATION:   6    ERROR: 1.80416E-02
     ITERATION:   7    ERROR: 7.42925E-03
     ITERATION:   8    ERROR: 1.27399E-02
     ITERATION:   9    ERROR: 1.58.09E-02
     ITERATION:  10    ERROR: 6.42318E-03
   WARNING: NO CONVERGENCE AFTER   10 ITERATIONS
   EXPERT DECISION: BACK OFF, ADJUST DT AND RETRY

   CURRENT TIME  =  1.2500000E-01     DT =  1.2500000E-01
     ITERATION:   1    ERROR: 1.00000E+00
     ITERATION:   2    ERROR: 2.14857E-02
     ITERATION:   3    ERROR: 7.79365E-04
     ITERATION:   4    ERROR: 1.28068E-05
   CONVERGENCE OBTAINED AFTER    4 ITERATIONS
   EXPERT DECISION: ADJUST DT AND CONTINUE
        .
        .
        .
        .
   CURRENT TIME  =  2.0000000E+00     DT =  2.8835625E-02
     ITERATION:   1    ERROR: 1.19598E-02
     ITERATION:   2    ERROR: 2.79259E-05
   CONVERGENCE OBTAINED AFTER    2 ITERATIONS

 REACHED TIME STOP =  2.0000000E+00
```

Figure 8.6: A copy of the echo of the nonlinear analysis of the wiper blade (only a few load steps are shown).



!! WARNING FROM THE VERIFICATION EXPERT SYSTEM !!

My analysis indicates that in your finite element analysis you are using constitutive equations valid only for small strains, while the strains reach level requiring large strain theory.

It is advised that you choose appropriate constitutive equations or change the model to reduce the strain level.

Figure 8.7: Deformed configuration of a wiper blade with stress intensity contours.

mance monitor expert system. Besides the features illustrated here, the system is equipped with bifurcation detection rules, protection from erroneous user data, and other additional verifications.

8.4 Adaptive Mesh Refinement

The adaptive mesh refinement procedure is designed to automatically adjust the finite element discretization in order to reduce the solution error at minimum cost. In this section an example of the *h-p* adaptive mesh refinement will be presented. This procedure was discussed in Section 6.4.1.

As a particular test problem an example of the finite element analysis of a wrench was selected. The finite element mesh for this tool is presented in Fig. 8.8. The length of the wrench is about 120mm and the material properties are the Young modulus $E = 210\ GPa$, Poisson's ratio $\nu = 0.3$, and the elastic limit stress $\sigma_E = 10\ MPa$. The interaction with the bolt or the nut head was modeled by appropriate displacement boundary conditions on the hex head and the torque was imposed by a distributed normal load on the handle.

The solution of this problem on the initial mesh in Fig. 8.8 gives the stress distribution presented in Fig. 8.9. Due to the stress concentration in the hex head zone, the error in this zone is much larger than elsewhere in the domain Ω and reaches a level up to 16 percent (see Fig. 8.10). To reduce the error level the *h-p* adaptive procedure was activated. The adaptive procedure was purely algorithmic and no expert system assistance was used. The final refined mesh is presented in Fig. 8.11. The contours of stress intensity and elementwise error indicators are presented in Figs. 8.12 and 8.13, respectively. For the sake of comparison, the contouring ranges for error and stress plots were the same as for the initial mesh. Note that the maximum local error dropped below three percent, while the maximum value of stress intensity was much higher than for the original mesh (this is due to stress concentrations on the hex head).

This simple example illustrates the powerful *h-p* adaptive mesh refinement capability. In more complex situations, an additional expert system will be used to assist the basic refinement procedure.

8.5 Adaptive Selection of Implicit and Explicit Zones

The adaptive implicit/explicit method, discussed in Section 6.3.4, will be illustrated on the example of the Mach 3 viscous flow about the flat plate at Reynolds number equal 1000. The detailed problem statement for this classic example can be found in reference [79]. This problem was solved with an implicit/explicit algorithm combined with adaptive

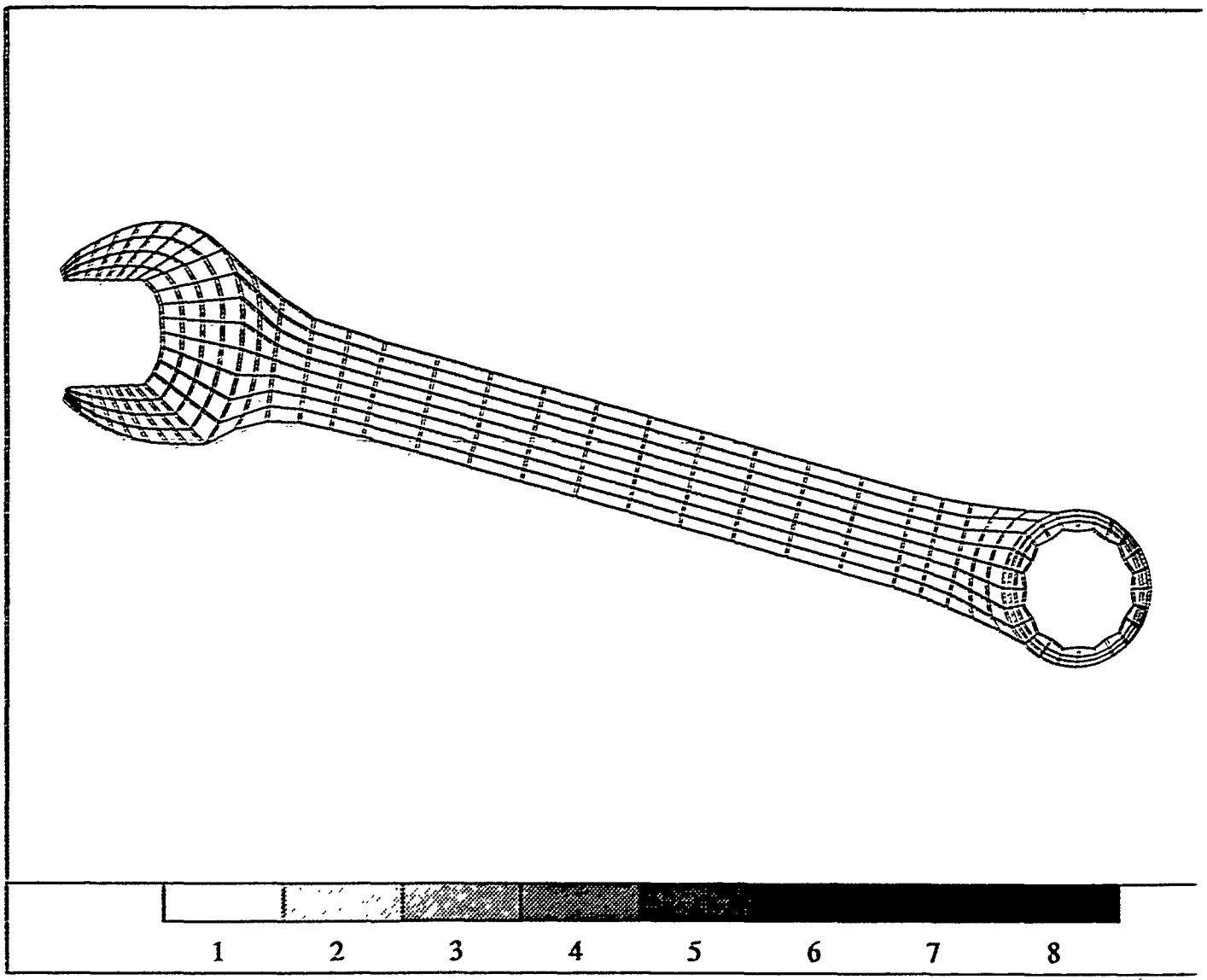


Figure 8.8: A finite element mesh for the wrench analysis. Shade intensity represents the order of approximation.

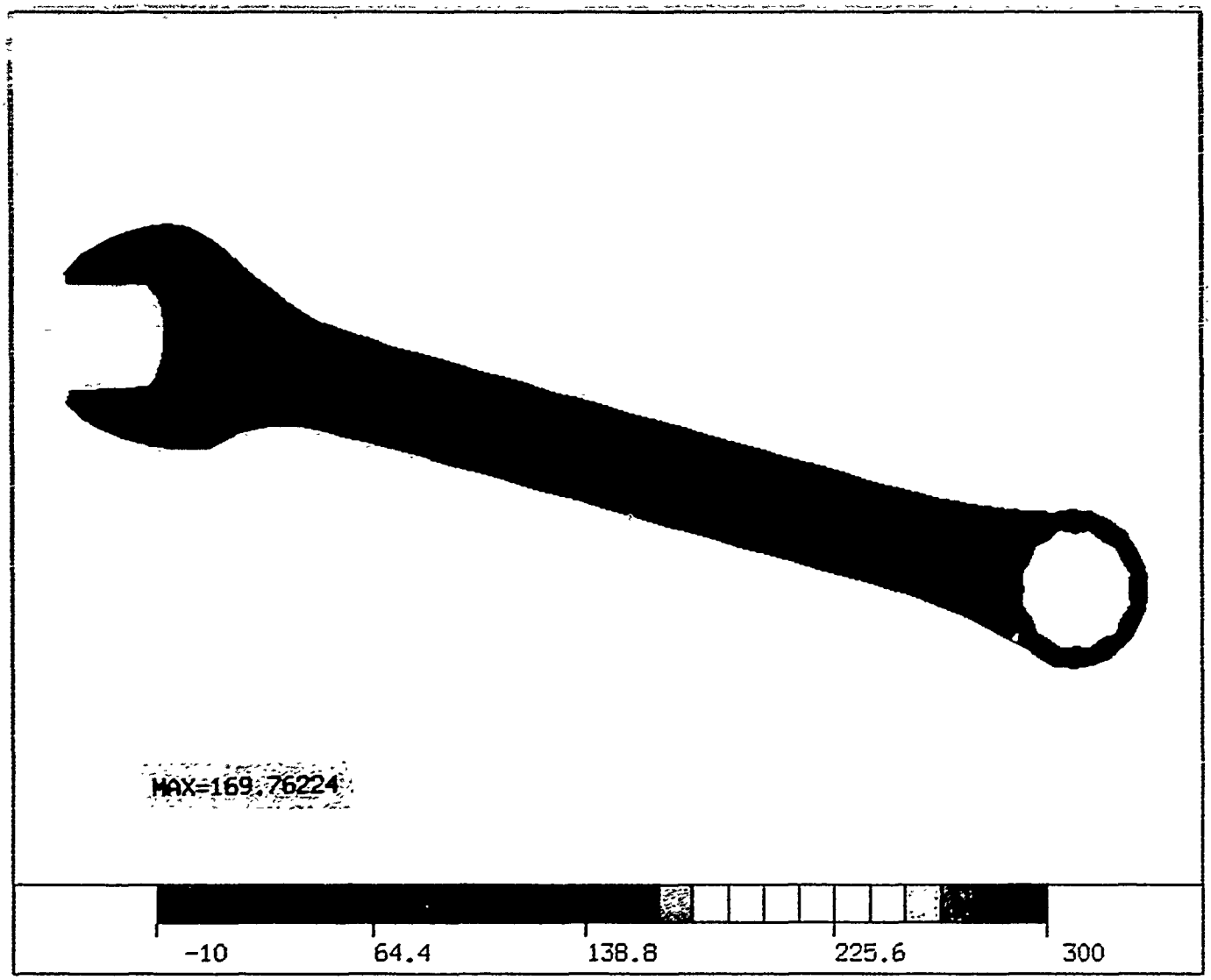


Figure 8.9: Stress contours for the wrench problem obtained at the initial mesh.

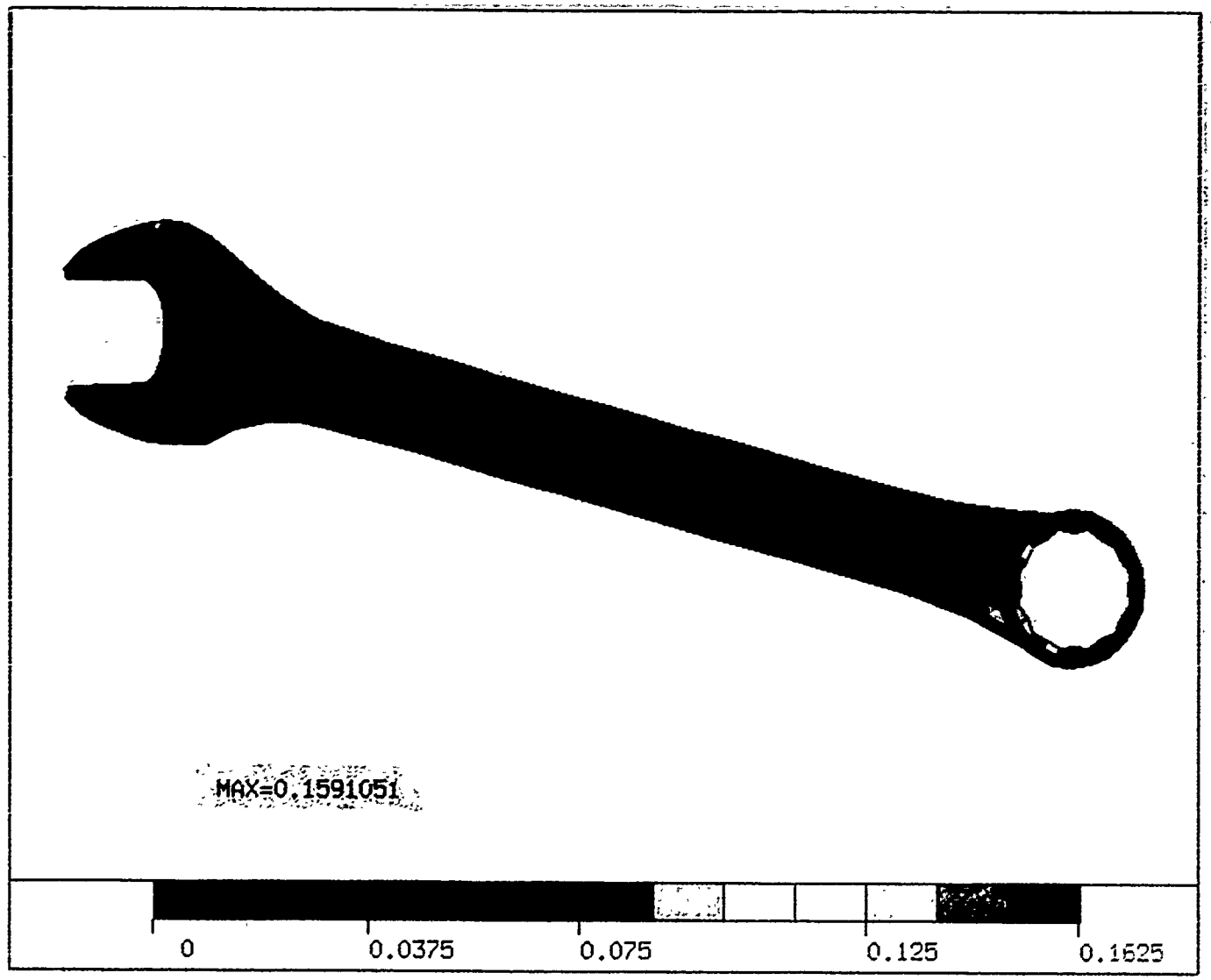


Figure 8.10: Error distribution for the wrench problem obtained at the initial mesh.

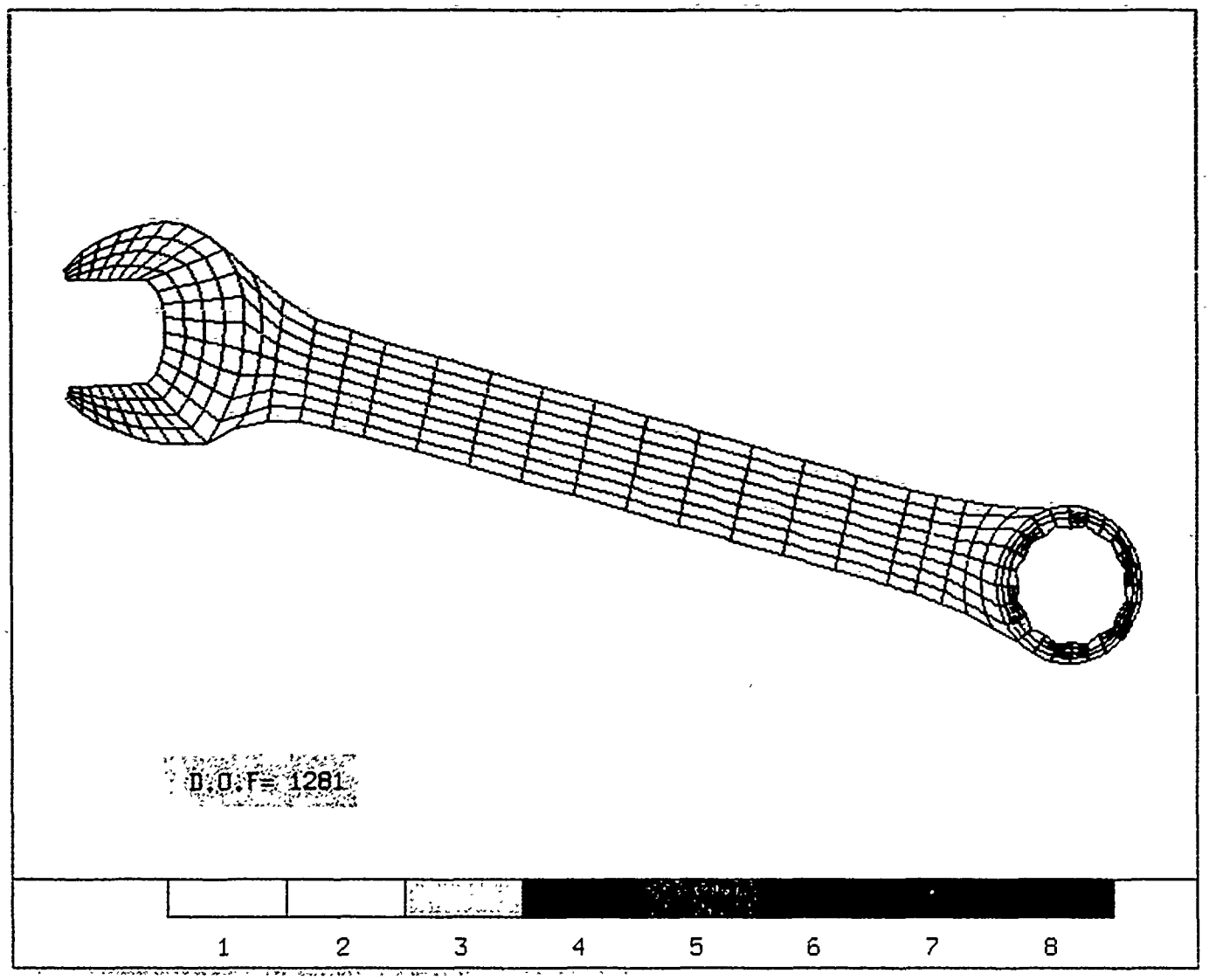


Figure 8.11: Adapted finite element mesh for the wrench analysis.

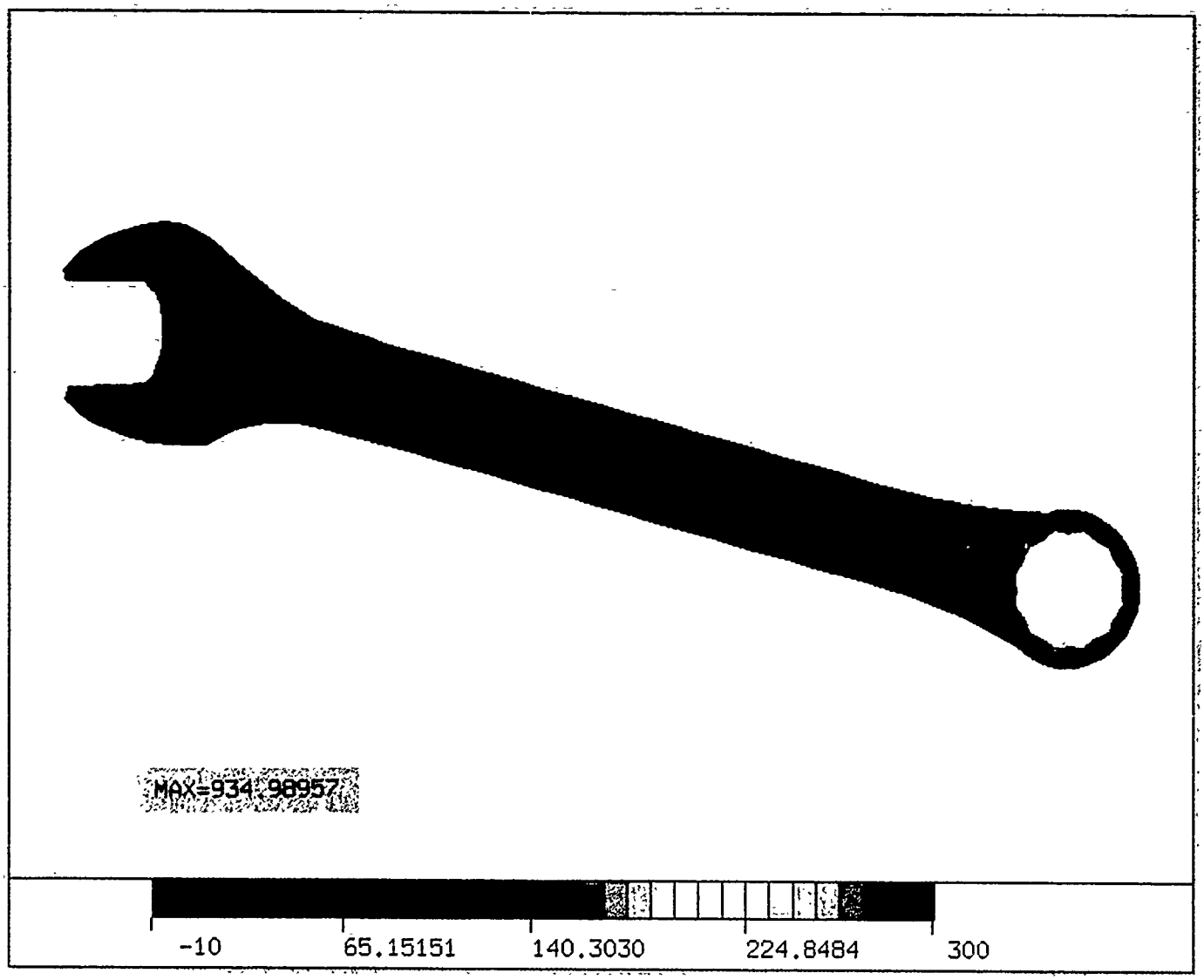


Figure 8.12: Stress contours for the wrench problem obtained at the adapted mesh.

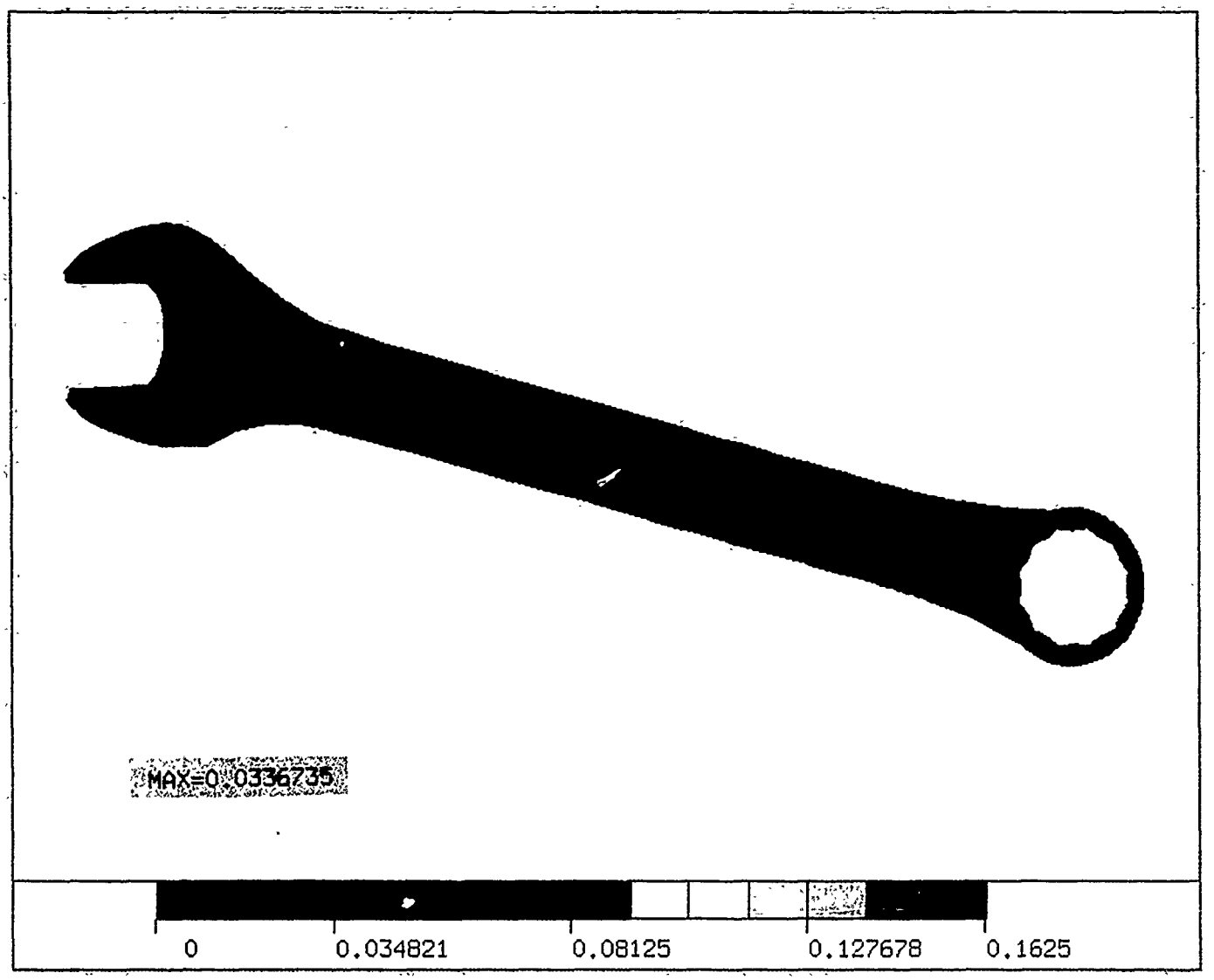


Figure 8.13: Error distribution for the wrench problem obtained at the adapted mesh.

mesh refinement. The selection of the implicit and explicit zones was based on the cost minimization. The criterion for the application of an implicit algorithm was a stability limitation of an explicit version. For the mesh presented in Fig. 8.14, the procedure, based on cost minimization, chose a time step about 20 times larger than the admissible time step for the fully explicit version. This was achieved by automatically selecting about 75 percent of the elements as implicit (1646 out of 2155). These implicit elements were clustered around the plate tip and in the boundary layer—see Fig. 8.14. The computational cost of reaching the steady–state solution was reduced by a factor of 2, as compared with the fully explicit algorithm.

The contours of density obtained for this problem are presented in Fig. 8.15. Note that, similarly as for adaptive mesh refinement, an expert system assistance can be used in more complex situations to improve the performance of the adaptive implicit/explicit procedure.

8.6 Automated Verification of Finite Element Results

The verification expert system is designed to automatically assure compliance of the finite element results with the basic design criteria, with the limitations of the mathematical model, etc. It is assumed that the approximation error has been reduced below a prescribed limit by the adaptive procedure and it is not of concern here.

The verification expert system is based on concepts discussed in Section 6. A general structure of the system is presented in Fig 8.16. A single object "versolid" represents formally a human expert working on the problem. It has several slots representing necessary information as well as associated rules representing the active expert knowledge. The actual solid objects are members of a class "solids." In this version only two–dimensional solid objects are considered. In general, additional classes of three–dimensional solids, plates, shells, beams, and other structural elements may be considered. The listing of classes, objects and rules of the verification expert system is presented in Appendix B3.

The automated verification is activated by selecting the command VERIFY in the post-processing menu of the finite element code. At the beginning of the verification session the class of solids is empty. Then, during the session, objects of this class are dynamically generated. These objects correspond to components of the structure. For each solid object the postprocessing of displacements, stress, strain, and other parameters is performed using methods developed in Section 6. The results of this postprocessing are the basis for the verification session of the expert system. In the cases of violation of the design criteria, improper selection of the mathematical model or other modeling errors, a relevant message is issued to the user together with suggested ways of fixing the problem. For more specialized applications, provisions can be made to automatically implement the necessary changes.

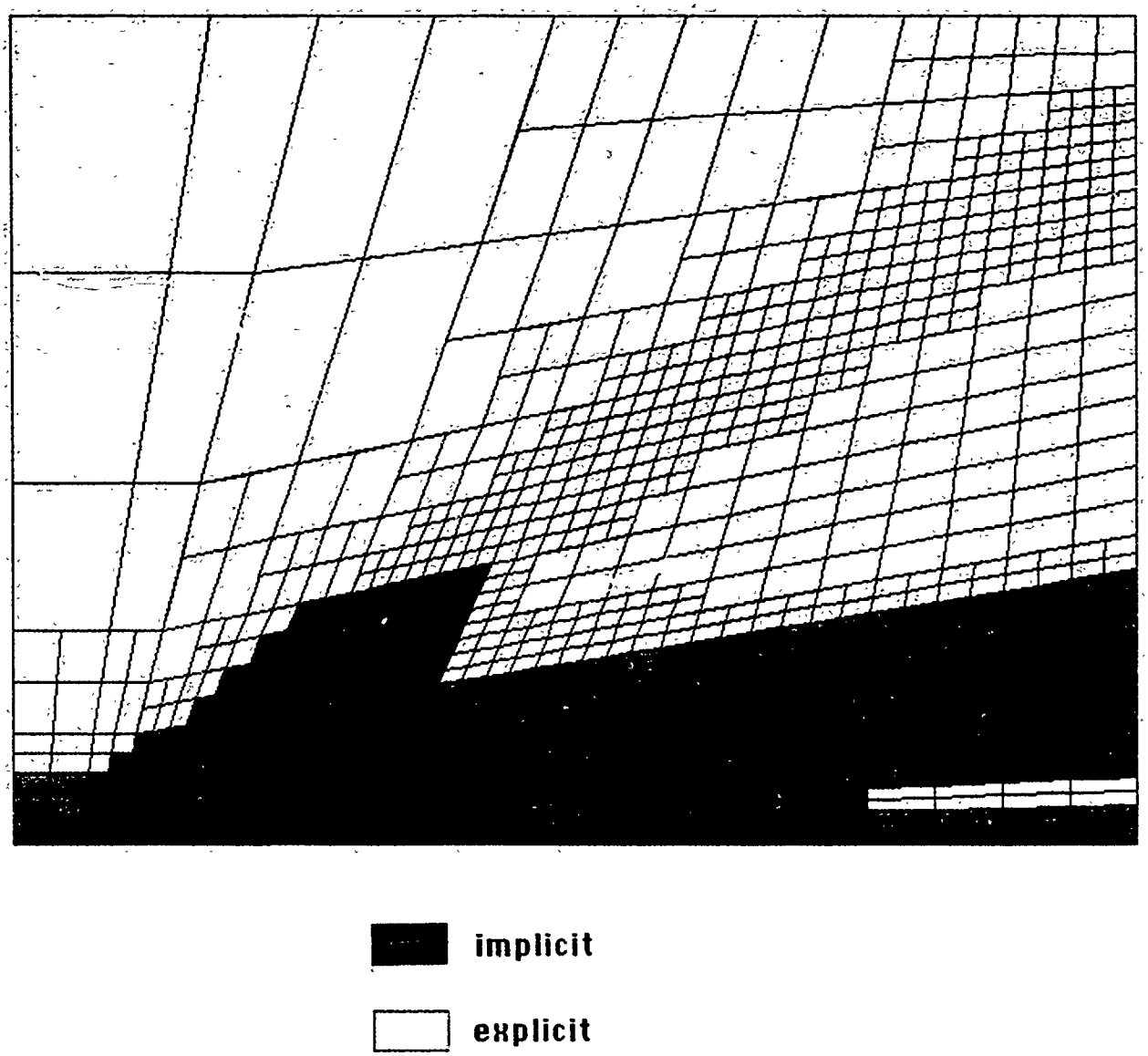


Figure 8.14: Automatically selected implicit and explicit zones for the flat plate viscous flow.

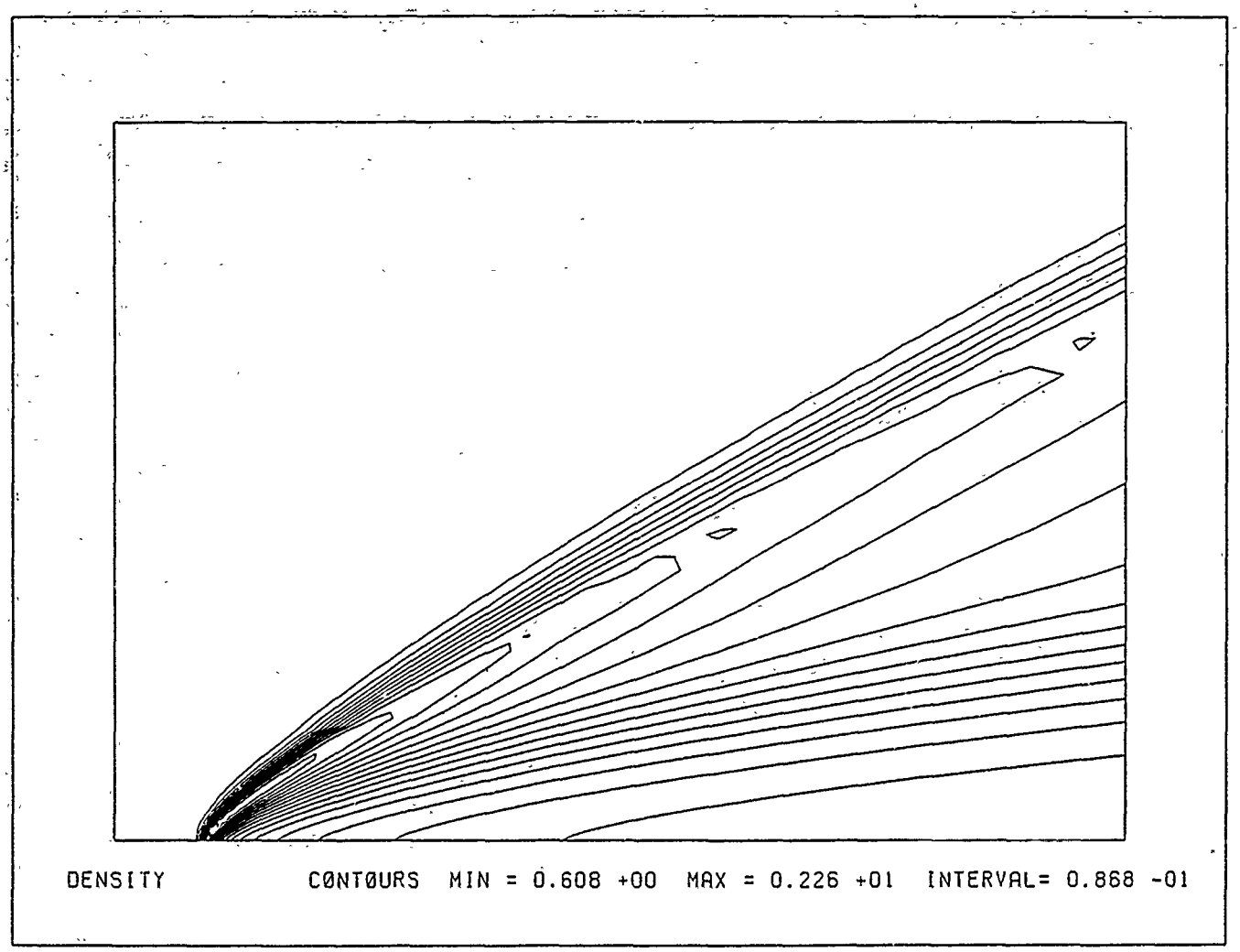


Figure 8.15: Density contours for the flat plate viscous flow.

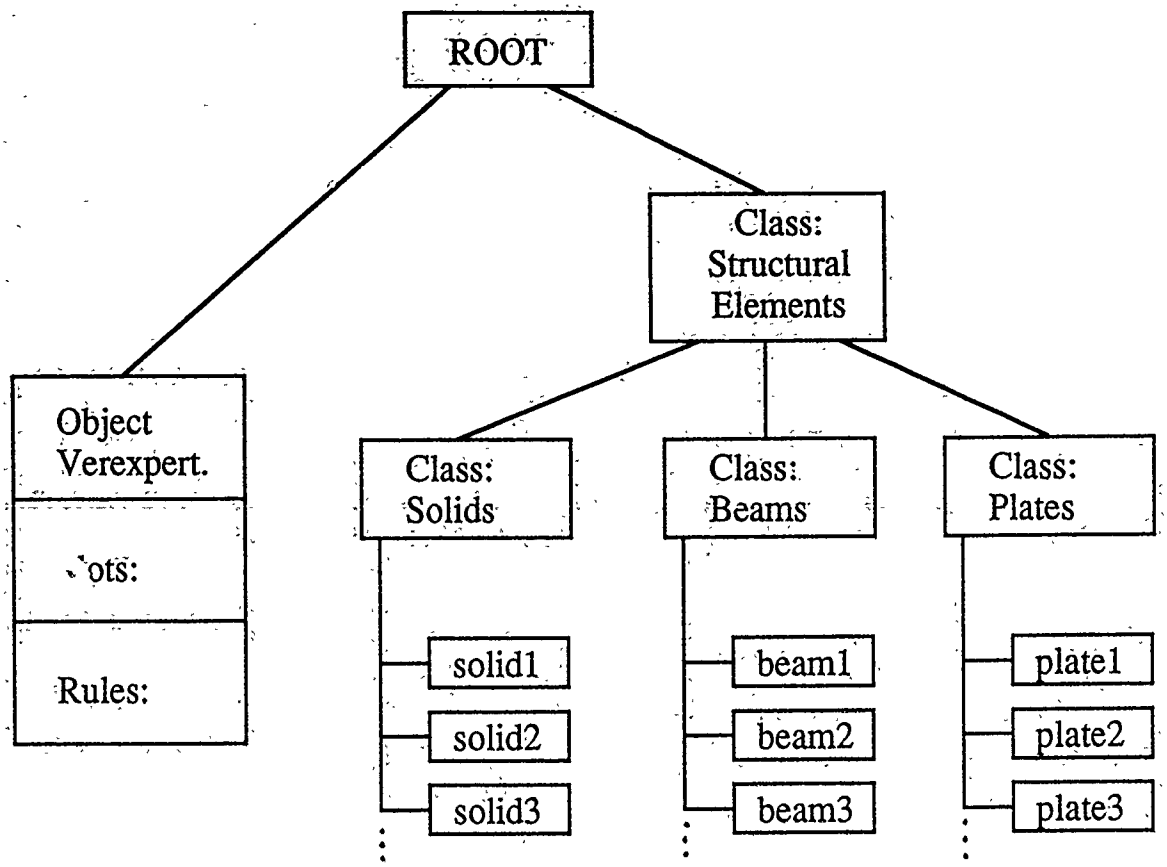


Figure 8.16: A general structure of the verification expert system.

8.6.1 Examples of Automated Verification of Results

The operational principles of the verification expert system will be illustrated in the example of the wrench analysis, introduced in the previous section. The process of finite element solution of this problem will be simulated here, including various modeling errors introduced by the user. For simplicity, no adaptive mesh refinement was used in the examples presented.

Case 1: Insufficient Supports

In the first run a common situation of improperly specified boundary conditions was simulated. In particular, no Dirichlet boundary conditions were specified, so that the wrench was subject to very large displacements and rotations. This situation was detected by the large displacement and rigid motion modules, and the expert system issued an appropriate message to the user. The deformed configuration and the message issued by the expert system are presented in Fig. 8.17.

Case 2: Excessive Loads

In this case the supports were properly defined, but the user specified excessive value of the traction load, so that the deformation of the wrench was large, see Fig. 8.18. This violated the assumptions of infinitesimal deformation theory in terms of both kinematics and constitutive equations. Thus this analysis resulted in two error messages, presented in Fig. 8.18. Note that the expert system properly concluded that the large rotation in this case was not rigid and was not caused by insufficient supports.

Case 3: Elastic Limit Exceeded

The loads specified in this case did not cause excessive deformations, yet were large enough to cause violation of the specified stress limits. However, the high stress occurs only in a small fraction of the domain, presented in Fig. 8.19. In general, these local concentrations may be essential for the design or can be caused by a relatively crude modeling of the actual shape (e.g., sharp corners introduced by the discretization). The expert system detected this stress concentration and issued the message to the user, accompanied by the list of possible solutions to this problem (Fig. 8.19).

Note that in more specific applications the conclusions of the expert system can be more specific and even an automated correction of the problem can be implemented.

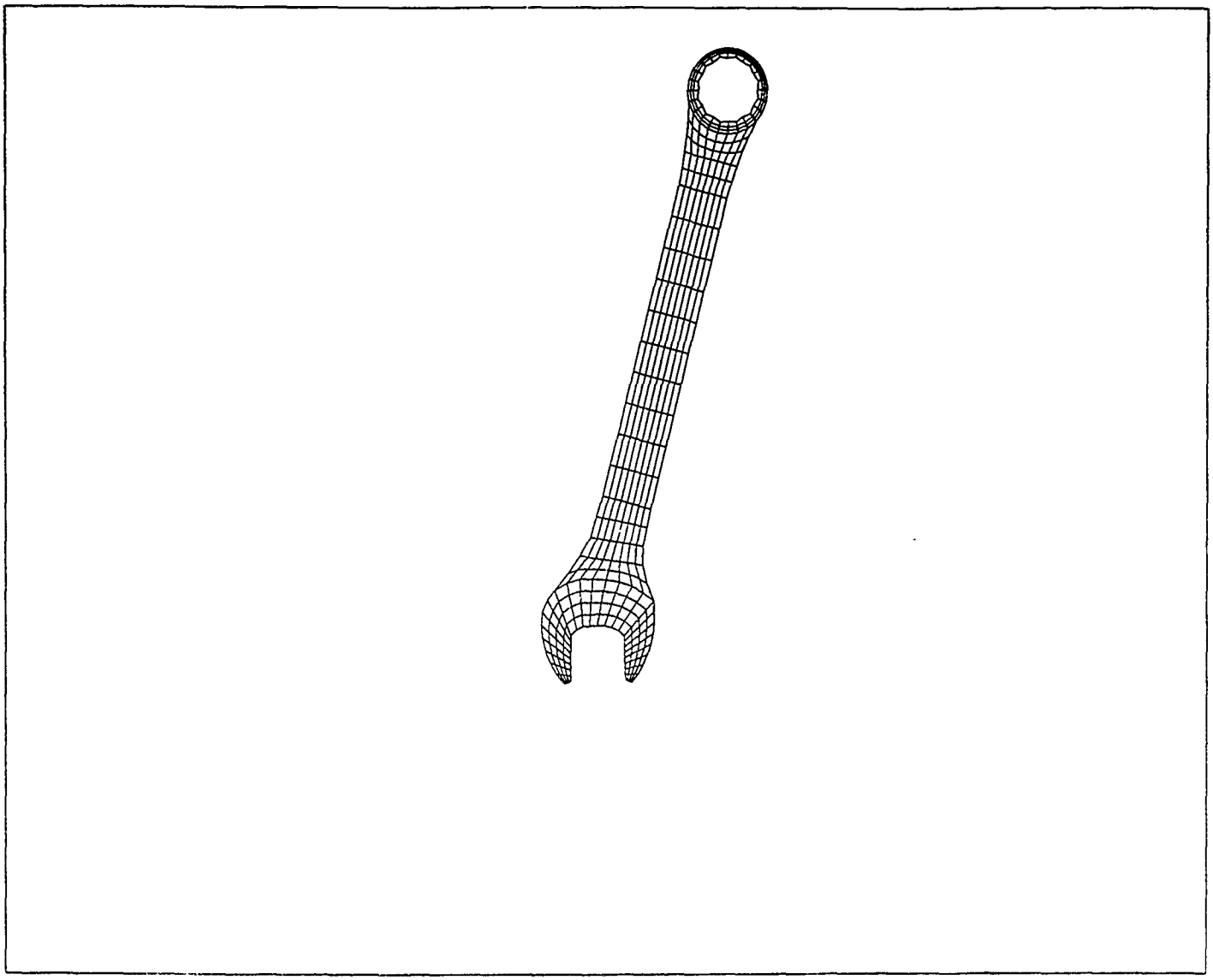
Case 4: Correct a Result

After reducing the load level on the handle, the deformation and stresses in the wrench were within the bounds of the infinitesimal deformation theory and below the prescribed stress limits. No warning messages were issued.

Case 5: Too Comprehensive a Theory Used

Here the situation was simulated when the user selected too comprehensive a mathe-

PROJECT: wrench1 DISPLACEMENTS



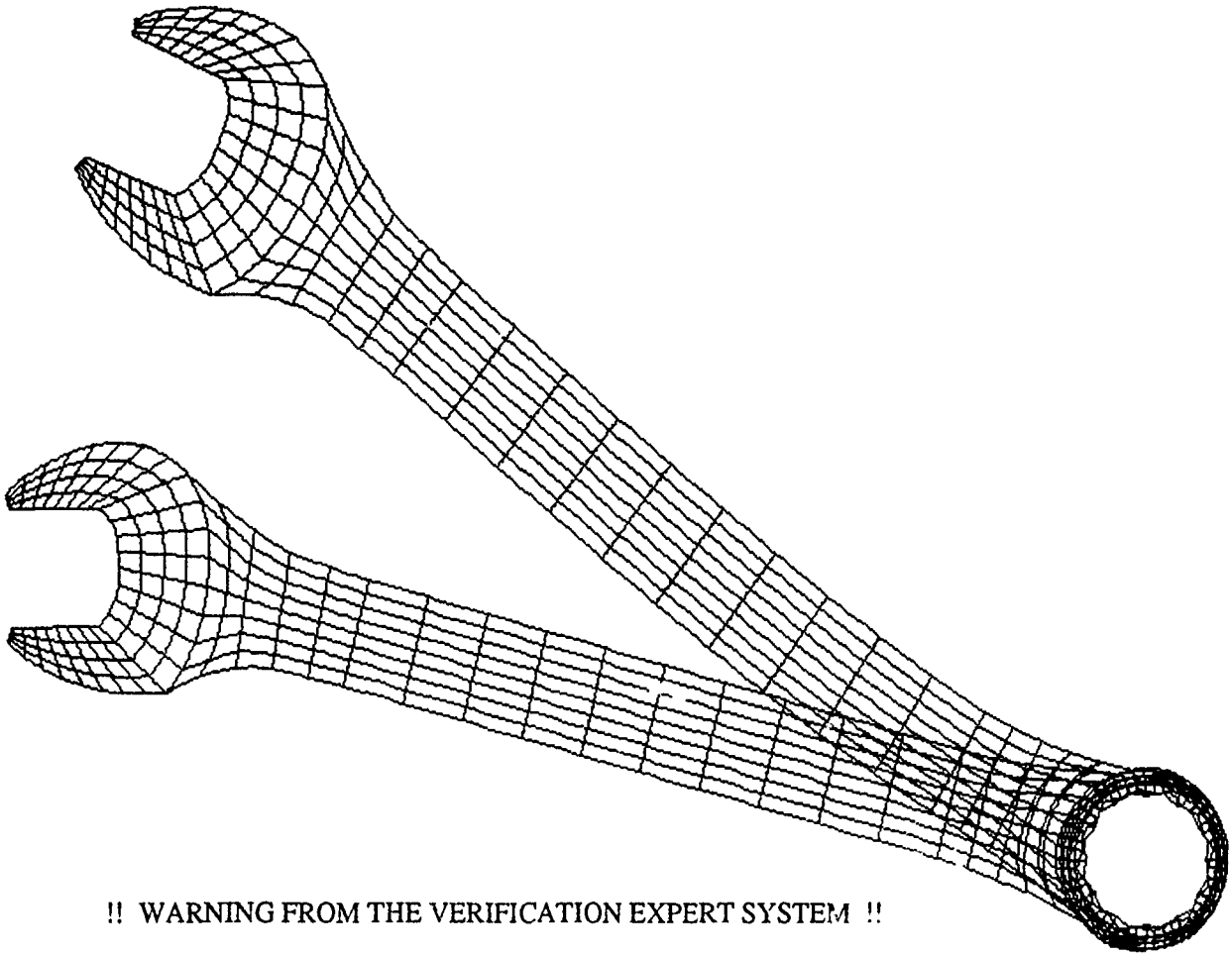
!! WARNING FROM THE VERIFICATION EXPERT SYSTEM !!

My analysis indicates that the structure or it's part undergoes excessive displacements and rotations. This is most probably caused by:

- lack of proper support (boundary conditions)
- missing connections between different parts

Unless you really accept the solution with such displacements and rotations, please examine and modify your data.

Figure 8.17: Deformed configuration and warning message in the case of insufficient supports.



!! WARNING FROM THE VERIFICATION EXPERT SYSTEM !!

My analysis indicates that in your finite element solution you are using infinitesimal deformation theory, while the large deformation theory is needed. This can lead to erroneus results.
Please switch to large displacement theory or, if you do not expect large displacements in your solution, check and modify:

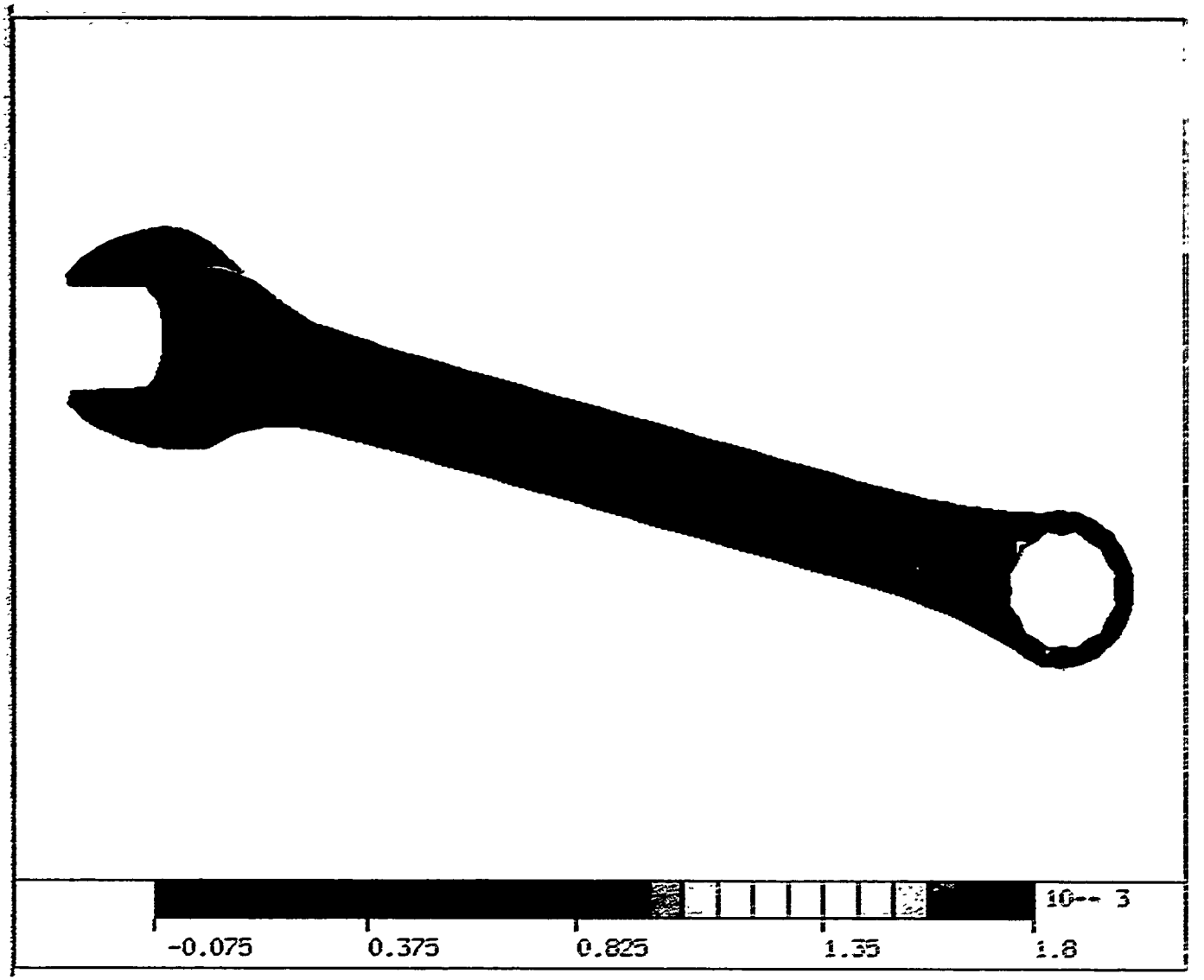
- material properties
- boundary conditions (loads)
- model geometry

!! WARNING FROM THE VERIFICATION EXPERT SYSTEM !!

My analysis indicates that in your finite element analysis you are using constitutive equations valid only for small strains, while the strains reach level requiring large strain theory.

It is advised that you choose appropriate constitutive equations or change the model to reduce the strain level.

Figure 8.18: Deformed configuration and warning message in the case of large rotations caused by excessive loads.



!! WARNING FROM THE VERIFICATION EXPERT SYSTEM !!

My analysis indicates that in the finite element analysis you are using elastic type constitutive equations, but the stresses exceed elastic limit.

However, these stresses occurr in less than 10 % of the area. Therefore you may chose to ignore them if this is a rather crude analysis. If not, then please:

1) modify the geometry of the model to avoid stress concentrations (e.g. round-off the corners) or
1) modify the model to reduce stress levels or
2) choose elasto-plastic constitutive theory (currently not available).

Figure 8.19: Stress contours and warning message in the case of violation of elastic stress limits.

```
 ENTER:   PROJECT_NAME
wrench4
solve
NONLINEAR PROCESS:
   ITERATION:  1   ERROR: 1.00000E+00
   ITERATION:  2   ERROR: 2.35496E-03
   ITERATION:  3   ERROR: 1.7?74?E-05
  CONVERGENCE OBTAINED AFTER  3 ITERATIONS

 FINISHED NONLINEAR PROCESS
end
post
verify

   !! MESSAGE FROM THE VERIFICATION EXPERT SYSTEM !!

 My analysis indicates that you are using a large deformation
theory while the deformations are in fact infinitesimal. Although
this is a correct and safe approach, you may want to use a simpler
and computationally cheaper infinitesimal deformation theory.

Session finished
end
```

Figure 8.20: Echo of the session with too comprehensive a mathematical model requested by the user.

matical model for the problem, which could effectively be solved using simpler theories. In particular, the wrench problem (Case 4) was analyzed using a large deformation theory and a nonlinear solution algorithm. This required more expensive computations than in the simple linear case. Thus the expert system issued a message to the user suggesting the possibility of switching to a simpler theory. The echo of this computational session is presented in Fig. 8.20.

Case 6: Incorrect Combination of Mathematical Theories

The last case considered here corresponds to a wiper blade analysis, discussed in Section 8.3. In this example, the kinematic theory was properly selected to be a large deformation theory. However, the constitutive equations were assumed to be linear, described by Hooke's law. Since the strains in the blade were actually large, application of Hooke's law was incorrect. Thus a warning message was issued by the expert system. This message is shown at the bottom of Fig. 8.7.

Note that the expert system checked whether the deformations were actually large. If the strains were within the range of small strains, the combination of linear constitutive theory with large deformation kinematics would have been accepted.

9 Conclusions

The theoretical analysis, literature survey, general design, and numerical tests performed in the project suggest that efforts toward automatic decision making in computational mechanics are definitely feasible and promise great payoff in practical applications. Once implemented and operational, the automated version will greatly improve quality, reliability, and time efficiency of the design by providing:

- **Guidance for the inexperienced engineer through the maze of engineering software, computational procedures and parameters.**
- **Assistance to the experienced designer in the selection of mathematical models, computational strategies, verification of the design, and iterative modification of the model.**
- **Automatic control of the quality of the computational results, in particular keeping the error within the prescribed limit.**
- **Minimizing the computational effort necessary to obtain good quality results by automated designing of an optimal mesh, optimal time step, load step, etc.**
- **Automated handling of computational difficulties (divergence, instability) based on the expertise of the program developer and experienced user.**
- **Automatic learning capabilities, with the system's own experience growing with the number of solved problems.**

The research performed in the project proves that developing an automated design environment is feasible and that it requires a combination of various computer tools and techniques, such as algorithmic procedures, CAD, computer graphics, and knowledge engineering. In particular, our work has shown that the techniques of knowledge engineering such as expert systems, are mature enough to be effectively used in automated computational mechanics. Nevertheless, it is our opinion that the potential of these systems should not be overestimated and that methods based on rigorous mathematical foundations and well defined procedures should be used whenever possible. Only in areas where such a precise knowledge does not exist or is not complete, a heuristic approach based on experience and expertise should be used.

It is essential that this heuristic knowledge in the automated environment is handled by appropriate tools of knowledge engineering, such as expert systems. Although it is possible to implement the expert rules in the form of classical if-then-else statements in the executable code, there are many arguments for application of expert system software for this purpose. The most important of these arguments are listed below.

1. **The implementation of rules in the form of if-then-else statements is formally possible, but it becomes extremely complex and time consuming for larger knowledge bases (say hundreds of rules). Moreover, maintenance and update of these rules is very difficult.**

2. **Today's advanced inference engines can navigate through the network of objects and rules in a more efficient fashion than the sequential approach of the if-then-else type. In particular, the same knowledge (rule) can be used for different purposes, depending on the current application of the expert system (induction, diagnostics, etc.).**

3. **In the knowledge engineering approach, the expert knowledge is stored in separate knowledge bases, rather than being hardwired into the code. This enables easy development and use of custom tailored expertise for different applications of the same generic software.**

4. **With new emerging techniques of artificial intelligence it can be expected that the automated environments using the knowledge engineering software will automatically benefit from the progress in this field and from new capabilities, such as knowledge acquisition, self-learning, etc.**

9.1 Directions of Future Work

The research performed during this project can be viewed as a feasibility study and proof of concept for the development of an automated environment in computational mechanics. Although the numerical implementations developed here were proven to be very useful and effective in practice, the development of the ultimate, fully automated environment for engineering design will require further intensive research. The basic directions of this research should include:

1. Further development of a rigorous mathematical background and heuristic knowlede for the automated selection of mathematical models of physical phenomena.

2. Automated generation of finite element models, to include full interaction of CAD modeling, mesh generation, and finite element software.

3. Continuous development and improvement of adaptive computational techniques and smart algorithms. This includes, in particular, *h-p* adaptive finite element methods, error estimation, adaptive time-sepping, adaptive zonal methods, etc.

4. Further work on automated assessment of the quality of numerical results and automated model verification.

5. Automated structural optimization, including a combination of algorithmic optimization methods with knowledge based support for qualitative changes in the design.

6. Development of an automated environment supporting a full cycle of the engineering design process, to include model generation by CAD and solid modeling, automated discretization, automated finite element analysis, computer aided manufacturing and automated cost analysis.

7. Continuous study of new computer technologies and methodologies, and their potential in the automated engineering design.

10 References

1. Anderson, J. A., "Data Representation in Neural Networks," *AI Expert*, 6, pp. 30–37, 1990.

2. Andrews, A. E., "Progress and Challenges in the Application of Artificial Intelligence to Computational Fluid Dynamics," *AIAA Journal*, Vol. 26, No. 1, pp. 40–46, 1988.

3. An-Nashif, H., and Powell, G. H., "A Strategy for Automated Modeling of Frame Structures," *Engineering With Computers*, 5, pp. 1–12, 1989.

4. Babuška, I., Gui, W., and Szabo, B., "Performance of the *h*, *p*, and *h-p* Versions of the Finite Element Method," Preprint, 1984.

5. Bayley, D., and Thompson, D., "How to Develop Neural Network Applications," *AI Expert*, 6, pp. 38–47, 1990.

6. Babuška, I., and Rank, E., "An Expert-System-Like Feedback Approach in *hp*-Version of the Finite Element Method," TN BN-1048, Institute for Physical Science and Technology, Lab. for Numerical Analysis, University of Maryland, 1986.

7. Babuška, I., and Miller, A., "The Post–Processing Approach in the Finite Element Method—Part 1: Calculation of Displacements, Stresses, and Other Higher Derivatives of Displacements," *Int. J. Numer. Methods Eng.*, 20, pp. 1085–1109, 1984.

8. Babuška, I., and Miller, A., "The Post–Processing Approach in the Finite Element Method—Part 2: The Calculation of Stress Intensity Factors," *Int. J. Numer. Methods Eng.*, 20, pp. 1111–1129, 1984.

9. Babuška, I., and Miller, A., "The Post–Processing Approach in the Finite Element Method—Part 3: *A Posteriori* Error Estimates and Adaptive Mesh Selection," *Int. J. Numer. Methods Eng.*, 20, pp. 2311–2324, 1984.

10. Bass, J. M., and Oden, J. T., "Adaptive Finite Element Methods for a Class of Evolution Problems in Viscoplasticity," *Int. J. Engng. Sci.*, 25 6, pp. 623–653, 1987.

11. Bathe, K. J., Lee, N., and Bucalem, M. L., "On the Use of Hierarchical Models in Engineering Analysis," Proc. of the Workshop on Reliability in Comp. Mech., Austin, Texas, pp. 5–27, October 1989, Ed. J. T. Oden, Elsevier, NY, 1990.

12. Bennett, J., Creary, L., Englemore, R., and Melosh, R., "A Knowledge–Based Consultant for Structural Analysis," Tech. Report STAN-CS-78-699, Stanford University, September, 1978.

13. Bermudez-Viaño, J. M., "Une Justification des Équations de la Thermo–Élasticité des Poutres à Section Variable par des Méthodes Asymptotiques," *RAIRO Analyse Numérique*, **18**, pp. 347–376, 1984.

14. Blacker, T. D., Mitchiner, J. L., Phillips, L. R., and Lin, Y. T., "Knowledge System Approach to Automated Two–Dimensional Quadrilateral Mesh Generation," *Computer in Engineering*, Vol. 2, 1988.

15. Bramble, J. H., and Schatz, A. H., "Higher Order Accuracy by Averaging in the Finite Element Methods," *Math. Comp.*, **31**, pp. 94–111, 1977.

16. Brauchli, H. J., and Oden, J. T., "Conjugate Approximation Functions in Finite Element Analysis," *Journal of Applied Mathematics*, April, 1971.

17. Braun, R., "Expert System Tools for Knowledge Analysis," AI Expert, October 1989, pp. 22–29.

18. Brezzi, F., "On the existence, Uniqueness and Approximatio of Saddle–Point Problems Arising from Lagrangian Multipliers," *RAIIRO Analyse Numérique*, Sèr. Rouge – 1, pp. 129–151, 1974.

19. Cagan, J., and Genberg, V., "PLASHTRAN: An Expert Consultant on Two–Dimensional Finite Element Modeling Techniques," *Engineering With Computers*, 2, pp. 199–208, 1987.

20. Caudill, M., "Using Neural Nets: Diagnostic Expert Nets," *AI Expert*, 7, pp. 43–47, 1990.

21. Caudill, M., "Using Neural Nets, Part 2: Fuzzy Decisions," AI Expert, April 1990, pp. 59–64.

22. Chen, J. L., and Haleja, P., "OPSYN: A CAD Based Expert System for Optimum Structural Synthesis," *Engineering Optimization*, Vol. 14, pp. 267–286, 1989.

23. Ciarlet, P. G., and Destuynder, P., "A Justification of the Two–Dimensional Linear Plate Model," *J. Mécanique*, Vol. 18, pp. 315–344, 1979.

24. Ciarlet, P. G., "Plates and Junctions in Elastic Multi–Structures, An Asymptotic Analysis," Springer-Verlag, 1990.

25. Craig, A. W., Zhu, J. A., and Zienkiewicz, O. D., "*A Posteriori* Error Estimation Methods Using Hierarchical Finite Element Bases," Inst. Num. Meth. Eng., C/R/483/84.

26. Demkowicz, L., "Some Remarks on Moving Finite Element Methods," *Computer Methods App. Mech. and Eng.*, Vol. 46, pp. 339–349, 1984.

27. Demkowicz, L., Oden, J. T., and Devloo, P., "An *h*-Type Mesh Refinement Strategy Based on a Minimization of Interpolation Error," *Comp. Meth. Appl. Mech. and Eng.*, Vol. 53, pp. 67–89, 1985.

28. Demkowicz, L., and Oden, J. T., "A Review of Local Mesh Refinement Techniques and Corresponding Data Structures in *h*-Type Adaptive Finite Element Methods," *TICOM Report*, 88-2, Austin, 1988.

29. Demkowicz, L., Oden, J. T., Rachowicz, W., and Hardy, O., "Toward a Universal *h-p* Adaptive Finite Element Solver," **TICOM Report**, The University of Texas at Austin, 1989.

30. Demkowicz, L., Oden, J. T., Rachowicz, W., and Hardy, O., "Toward a Universal *h-p* Adaptive Finite Element Strategy, Part I: Constrained Approximation and Data Structure," *Comp. Meth. Appl. Mech. Engng.*, 77, pp. 79–112, 1989.

31. Farouki, R. T., and Hinds, J. K., "An Hierarchy of Geometric Forms," *IEEE Computer Graphics and Applications 5* No. 5, pp.51–78, May, 1985.

32. Fenves, S. J., "A Framework for a Knowledge–Based Finite Element Assistant," ASME, Special Publication AD-10, New York, 1985.

33. Foley, J. D., and Van Dam, A., "Fundamentals of Interactive Computer Graphics," Addison Wesley, Reading, MA, 1982.

34. Gregory, B. L., and Shephard, M. S., "Design of a Knowledge–Based System to Convert Airframe Geometric Models to Structural Models," **Expert Systems in Civil Engineering**, ASCE, New York, 1986.

35. Guan, J., and Ohsuga, S., "An Intelligent Man–Machine System Based on KAUS for Designing Feedback Control Systems," in **Artificial Intelligence in Engineering Design**, J. S. Gero (ed.), Elsevier Science Pub. Co., 1988.

36. Gurtin, M. E., "An Introduction to Continuum Mechanics," Academic Press, 1981.

37. Harvey, J. J., "Expert Systems: an Introduction: *Int. J. Comp. in Technology,* Vol, 1, Nos. 1 and 2, pp. 53–50.

38. Herbst, B. M., Mitchell, A. R., and Schoombie, S. W., "A Moving Petrov-Galerkin Method for Transport Equations," *Int. J. Numer. Meth. Eng.*, 18 pp. 1321–1336, 1982.

39. Herbst, B. M., Mitchell, A. R., and Schoombie, S. W., "Equidistributing Principles in Moving Finite Element Methods," *J. Comp. and Appl. Math.*, Vol. 9, No. 4, pp. 377–389, 1983.

40. Kaminski, J. S., Clitherow, P. A., and Stuk, G. J., "Integrating Expert Systems with Conventional Applications," *Computers in Engineering*, Vol. 1, pp. 225–229, 1988.

41. Kumar, V., Majaria, M., and Mukherjee, S., "Numerical Integration of Some Constitutive Models of Inelastic Deformation," *J. Engrg. Mat. and Tech.*, **102** pp. 92–96, 1980.

42. Lawrence, J. J., "Untangling Neural Nets," *Dr. Dobb's Journal*, **163**, pp. 38–44, 1990.

43. Lerner, E. J., "Computers that Learn," *Aerospace America*, June 1988, pp. 32–40.

44. Lions, J. L., "Perturbations Singulières dans les Problèms aux Limites et en Contrôe Optimal," **Lect. Notes in Mathematics**, Vol. 323, Springer-Verlag, Berlin, 1973.

45. Liszka, T., and Orkisz, J., "The Finite Difference Method at Arbitrary Irregular Grids and its Applications in Applied Mechanics," *Comp. and Struct.*, **11**, pp. 83–95, 1980.

46. Malvern, L. E., "Introduction to the Mechanics of a Continuous Medium," Prentice-Hall, 1969.

47. Meyer, B., **Object–Oriented Software Construction**, Prentice Hall, 1988.

48. Miller, K., "Recent Results on Finite Element Methods With Moving Nodes," ARFEC, Lisbon, 1984.

49. Miller, K., and Miller, R. N., "Moving Finite Elements, I," *SIAM Journal of Numerical Analysis*, Vol. 18, No. 6, 1981.

50. Muleller, A. C., and Carey, G. F., "Continously Deforming Finite Element Methods," *Int. J. Num. Meths. Eng.*, Vol. 21, Nos. 11, 12, pp. 2099–2130, 1985.

51. Noor, A. K., and Babuška, I., "Quality Assessment and Control of Finite Element Solutions, Finite Element in Analysis and Design," 1986 (to appear).

52. Oden, J. T., and Demkowicz, L., "Adaptive Finite Element Methods for Complex Problems in Solid and Fluid Mechanics," *Computational Mechanics*, ITT, Bombay, India, 1985.

53. Oden, J. T., Demkowicz, L., Rachowicz, W., and Westermann, T. A., "Toward a Universal *h-p* Adaptive Finite Element Strategy, Part II: *A Posteriori* Error Estimation," *Comp. Meth. Appl. Mech. Engng.*, 77, pp. 113–180, 1989.

54. Oden, J. T., Strouboulis, T., and Devloo, P., "Adaptive Finite Element Methods for Compressible Flow Problems," **Finite Element Methods in Compressible Flow**, Edited by T. E. Tezduyar, AMD Monograph, 1986.

55. Oden, J. T., Demkowicz, L., Strouboulis, T., and Devloo, P., "Adaptive Methods for Problems in Solid and Fluid Mechanics," **Accuracy Estimates and Adaptive Refinements in Refinement Element Computations**, Edited by Babuška, et al., John Wiley and Sons, Ltd., Chichester, 1986.

56. Oden, J. T., and Demkowicz, L., "Survey of Adaptive Computational Methods in State-of-the-Art Surveys in Computational Mechanics," ASME (to appear).

57. Ohsuga, S., "Towards Intelligent CAD Systems," *Computer-Aided Design*, Vol. 21, No. 5, pp. 315–337, June, 1989.

58. Oliveira, E. R., Arantes, "Optimization of Finite Element Solutions," Proc. of the Third Conf. on Matrix Methods in Struct. Mech., Wright-Patterson AFB, Ohio, October, 1971.

59. Powell, G. H., and An-Nashif, H., "Automated Modeling for Structural Analysis," *Engineering With Computers*, 4, pp. 173,–183, 1988.

60. Rachowicz, W., "An Evolution and Comparison of Post-Processing Methods for Finite Element Solution of Elliptic Boundary-Value Problems," **TICOM Report** 87-11, The University of Texas at Austin, 1987.

61. Rachowicz, W., Oden, J. T., and Demkowicz, L., "Toward a Universal *h-p* Adaptive Finite Element Strategy, Part III: Design of *h-p* Meshes," *Comp. Meth. Appl. Mech. Engng.*, **77**, pp. 181–212, 1989.

62. Rehak, D. R., "Artificial Intelligence Based Techniques for Finite Element Program Development," **Reliability of Methods for Engineering Analysis**, Swansea, U.K., Pineridge Press, pp. 515–532, 1986.

63. Requicha, A. A. G., and Voelcker, H. B., "Solid Modeling: A Historical Summary and Contemporary Assessment," *IEEE Computer Graphics and Applications 2*, No. 2, pp. 9–24, 1982.

64. Requicha, A. A. G., and Voelcker, H. B., "Solid Modeling: Current Status and Research Directions," *IEEE Computer Graphics and Applications 3*, No. 7, pp. 25–37, 1983.

65. Rogers, D. F., and Adams, J. A., "Geometric Modeling—Ten Years On," CAD Group Document No. 103, University of Cambridge, Computer Laboratory, Cambridge, U.K.

66. Sapidis, N., and Perruchio, R., "Advanced Techniques for Automatic Finite Element Meshing From Solid Models," *CAD*, Butterworths, Vol. 21, No. 4, pp. 248–253, May, 1989.

67. Shephard, M. S., Yerry, M. A., "Toward Automated Finite Element Modeling," *Finite Elements Anal. Des.*, **2**, pp. 143–160, 1986.

68. Shephard, M. S., "Approaches to the Automatic Generation and Control of Finite Element Meshes," *Appl. Mech. Rev.*, **41**, pp. 169–185, 1988.

69. Shephard, M. S., Korngolg, E. V., and Wentorf, R., "Design Systems Supporting Engineering Idealization," in **Geometric Modeling for Product Engineering**, North-Holland, Amsterdam, pp. 279–300, 1990.

70. Smithers, T., "AI Based Design Versus Geometry Based Design, or Why Design Cannot be Supported by Geometry Alone," *Computer Aided Design*, Vol. 21, No. 3, April, 1989.

71. Szabo, B. A., "Mesh Design for the p-Version of the Finite Element Method," *Comp. Meths. Appl. Mech. Engrg.*, Vol. 55, Nos. 1, 2, 1986.

72. Szabo, B. A., Basu, P. K., and Rossow, M. P., "Adaptive Finite Element Analysis Based on p-Convergence," NASA Conf. Pub. 2059, pp. 43–50, 1978.

73. Szabo, B. A., and Babuška, I., "Stress Approximations by the h- and p-Versions of the Finite Element Method," **Report WU/CCM-82/1**, Center for Computational Mechanics, Washington University, March, 1982.

74. Thornton, E. A., Oden, J. T., Tworzydlo, W. W., and Youn, S. K., "Thermo–Viscoplastic Analysis of Hypersonic Structures Subjected to Severe Aerodynamic Heating," *Journ. of Aircraft*, **27**, 9, pp. 826–836, 1990.

75. Thornbrugh, A. L., "Organizing Multiple Expert Systems: A Blackboard–Based Executive Application," in Knowledge Based Expert Systems for Engineering, eds. D. Sciram and R. A. Adey, Computational Mechanics Publications, 1987.

76. Tong, S. S., "Design of Aerodynamic Bodies Using Artificial Intelligence/Expert System Technique," AIAA Paper 85-0112, Jan., 1985.

77. Tong, S. S., "Coupling Artificial Intelligence and Numerical Computation for Engineering Design," AIAA Paper 86-0242, Jan., 1986.

78. Trabucho, L., and Viaño, J. M., "Derivation of Generalized Models for Linear Elastic Beams by Asymptotic Methods," *Applications of Multiple Scaling in Mechanics*, Edited by P. G. Ciarlet and E. Sanchez-Palencia, Masson, Paris, pp. 302–315, 1987.

79. Tworzydlo, W. W., Oden, J. T., and Thornton, E. A., "Adaptive Implicit/Explicit Finite Element Method for Compressible Viscous Flows," submitted to *Comp. Meth. Appl. Mech. Engng.*, 1991.

80. Tworzydlo, W. W., Oden, J. T., and Bass, J. M., "Non-Algorithmic Issues in Computational Mechanics," Annual Technical Report, AFOSR Contract F49620-89-C-0015, April 1990.

81. Wathen, A. J., Baine, M. J., and Morton, K. W., "Moving Finite Element Methods for the Solution of Evolutionary Equations in One and Two Space Dimensions," MAFELAP, 1984.

82. Weiler, K. J., "Topological Structures for Geometric Modeling," Ph.D. Thesis, TR 86039, Troy, New York, Center for Interactive Computer Graphics, RPI, 1986.

83. Zienkiewicz, O. C., and Craig, A. W., "Adaptive Mesh Refinement and *A Posteriori* Error Estimation for the *p*-Version of the Finite Element Method," in **Adaptive Computational Methods for Partial Differential Equations**, Ed. I. Babuška, et al., SIAM Publications, Philadelphia, PA, 1983.

84. Zlamal, M., "Superconvergence and Reduced Integration in the Finite Element Method," *Math. Comp.*, **32**, pp. 663–685, 1978.

APPENDICES

APPENDIX A

A An Adaptive *h-p* Finite Element Method for Two–Dimensional Problems

In this section we briefly present a system supporting simultaneous *h-p* refinements for an Adaptive Finite Element Method (AFEM) for two-dimensional problems of solid mechanics. This system offers the possibility of application of automatic decision making to obtain an optimal finite element mesh distribution and control of the error of the solution. These adaptive strategies, based on data structure presented here, are discussed in Section 6.2.

To represent the basic ideas of the *h-p* adaptive finite element method, consider a system of linear equations:

$$\begin{cases} \text{Find } \boldsymbol{u}_h \in \boldsymbol{X}_h & \text{such that} \\ a(\boldsymbol{u}_h, \boldsymbol{v}_h) = L(\boldsymbol{v}_h) & \forall \boldsymbol{v}_h \in \boldsymbol{X}_h \end{cases} \tag{A.1}$$

where

- $\boldsymbol{X}_h = \boldsymbol{X}_h x \ldots x \boldsymbol{X}_h$ (n times), with $\boldsymbol{X}_h$ being a finite element space corresponding to an adaptively changing irregular FE mesh consisting of quadrilaterals of locally varying size and order of approximation,
- $a(\cdot,\cdot)$ is a bilinear form defined on $\boldsymbol{X}_h \times \boldsymbol{X}_h$,
- $L(\cdot)$ is a linear form defined on $\boldsymbol{X}_h$.

Note the fact that the same approximation is used for each component of $u_h = (u_{h_1},\ldots,u_{h_n})$. The bilinear form $a(\cdot,\cdot)$ is assumed to have the following form

$$\begin{aligned} a(u,v) &= \int_\Omega \sum_{K,L=1}^{n} \left\{ \sum_{i,j=1}^{2} a_{KL}^{ij} \frac{\partial u_K}{\partial x_i}\frac{\partial v_L}{\partial x_j} + \sum_{i=1}^{2} b_{KL}^{i} \frac{\partial u_K}{\partial x_I} v_L + c_{KL} u_K v_L \right\} dx \\ &+ \int_{\partial\Omega} \sum_{K,L=1}^{n} d_{KL} u_K v_L \quad ds \end{aligned} \tag{A.2}$$

and the linear functional $L(v)$ takes the form

$$\begin{aligned} L(v) &= \int_\Omega \sum_{K=1}^{n} \left\{ f_K v_K + \sum_{i=1}^{2} g_K^i \frac{\partial v_K}{\partial x_I} \right\} dx \\ &+ \int_{\partial\Omega} \sum_{K=1}^{n} h_K v_K \quad ds \end{aligned} \tag{A.3}$$

Here $\Omega \subset R^2$ is a two-dimensional domain (replaced in practice with its FE approximation) with boundary $\partial\Omega$. Coefficients $a^{ij}_{KL}, b^i_{KL}, c_{KL}, d_{KL}, f_K, g^i_K, h_K$ are given functions specified in Ω or the boundary $\partial\Omega$, respectively.

Let us notice at this point that the approximations for both the solution u_h and the test function v_h are the same. This in particular implies that the essential boundary conditions must be coded in using the penalty method.

This presentation consists of three major parts. In the following section we describe the basic data structure. Section A.2 discusses the mesh modification algorithms while Section A.3 is devoted to a detailed presentation of the notion of the constrained approximation.

A.1 The *h-p* Data Structure

We shall adopt the following assumptions:

- The original mesh is (topologically) a portion of a regular, rectangular mesh. In particular all nodes in the mesh are regular and every element has up to four neighbors (elements adjacent to the boundary have less neighbors).
- Every element may have up to nine nodes: four vertices, four midpoints of the element edges and a central node. The geometry of the element is uniquely prescribed by the coordinates of these nodes.
- During a refinement/unrefinement process, irregular meshes of order 1 are accepted.
- There may be many degrees of freedom associated with one node.

A.1.1 Nodes and Degrees of Freedom

Node coordinates are stored in the standard way in an array

XNODE(2,·)

Since, every element may have up to 9 nodes, it is assumed that by using these nodes, the element can be deformed resulting in the subparametric deformation spanned by, at most, biquadratic functions.

The variable number of degrees of freedom associated with a node suggests that we must store all values of degrees of freedom in a sequential mode. This is done by using two arrays:

- A real array U(·) storing the degrees of freedom, and

- an integer array NADRES (-) storing addresses of first degrees of freedom corresponding to node.

More precisely, if L = NADRES(NODE) then U(L) contains the value of the first degree of freedom corresponding to node number NODE. During the refinement/unrefinement process, the number of degrees of freedom corresponding to a node may be changed, the node may be deleted, or a new node may appear. Thus, an amount of storage allocated for degrees of freedom corresponding to a node must vary in the program.

A.1.2 Connectivity

To represent element connectivities, we introduce the array

NODES(9, -)

which contains for every *active* element up to nine *nicknames* of the form

NICKNAME = NODE * 100 + NORDER

where NODE is the node number and NORDER stands for the order of approximation associated with the node. When an element is h refined, its 'first-born' son takes on its place in NODES, while only the next three sons are assigned a new allocation in the array.

A.1.3 The Tree Information

We store the tree information in a condensed, family-like fashion. If NRELEI denotes the number of the elements in the inital mesh, the integer array

NSONS(NRELEI)

is introduced with each refined element of the initial mesh containing the number of the first-born son. Since the next three sons take on the next three consequtive numbers, NSONS array allows us to determine all sons for the elements of the initial mesh.

Whenever an element is refined, a family is being created. We store information about the families in the integer array

NTREE(5, -)

For a K-th family, NTREE(1,K) contains the number of the 'father', and the next four entries are reserved for first-born sons of the sons of the father, shortly the first-born grandsons of the 'father'. When elements are being refined *only* the new families are created in a consequtive manner. If however, unrefinements take place, some of the families are deleted and new families take the place (and number) of the first free family entry.

A.1.4 Natural Order of Elements

Since new elements are created in a rather random way, and non-active elements preserve their numbers, the natural question arises, 'How can one place all the active elements in order?' In the code we propose the so-called *natural order of elements* based on the initial mesh numbering and the tree information. Instead of defining the order formally we present in Fig. A.1 a typical tree-structure for an initial mesh consisting of three elements. The arrows indicate the natural order of elements. Of course, the non-active elements are dropped in the order.

Let us finally mention that, if the elements in the initial mesh are well numbered then the natural order of elements guarantees, at least up to a certain extent, a minimal band width in the global stiffness matrix.

A.2 Mesh Modification Algorithms

A.2.1 p-Enrichments and p-Unenrichments

Two typical situations are depicted in Fig. A.2 If the modified element is of first order, new nodes are simply added with a number of degrees of freedom corresponding to the required order of approximation. In the case of a 'big' neighbor, the new node is added as a midpoint of the big neighbor edge (geometrically it coincides with one of the vertices of the element, comp. top drawing in Fig. A.2) and as a result of the constrained approximation the new shape functions are added to three elements simultaneously: the element, the 'big' neighbor and the neighbor (of the same size) sharing the same edge with the big element. As a result of a higher order of approximation of a neighbor, the element may already have some nodes corresponding to the higher order of approximation. In such a case these nodes are only modified and the new necessary nodes are added.

A decrease of the order of approximation is done in exactly reversed order. The nodes are modified or deleted.

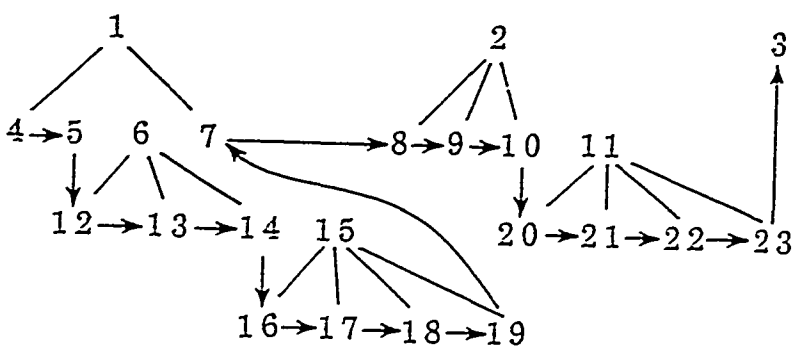


Figure A.1: The natural order of elements: 4, 5, 12, 13, 14, 16, 17, 18, 19, 7, 8, 9, 10, 20, 21, 22, 23, 3.

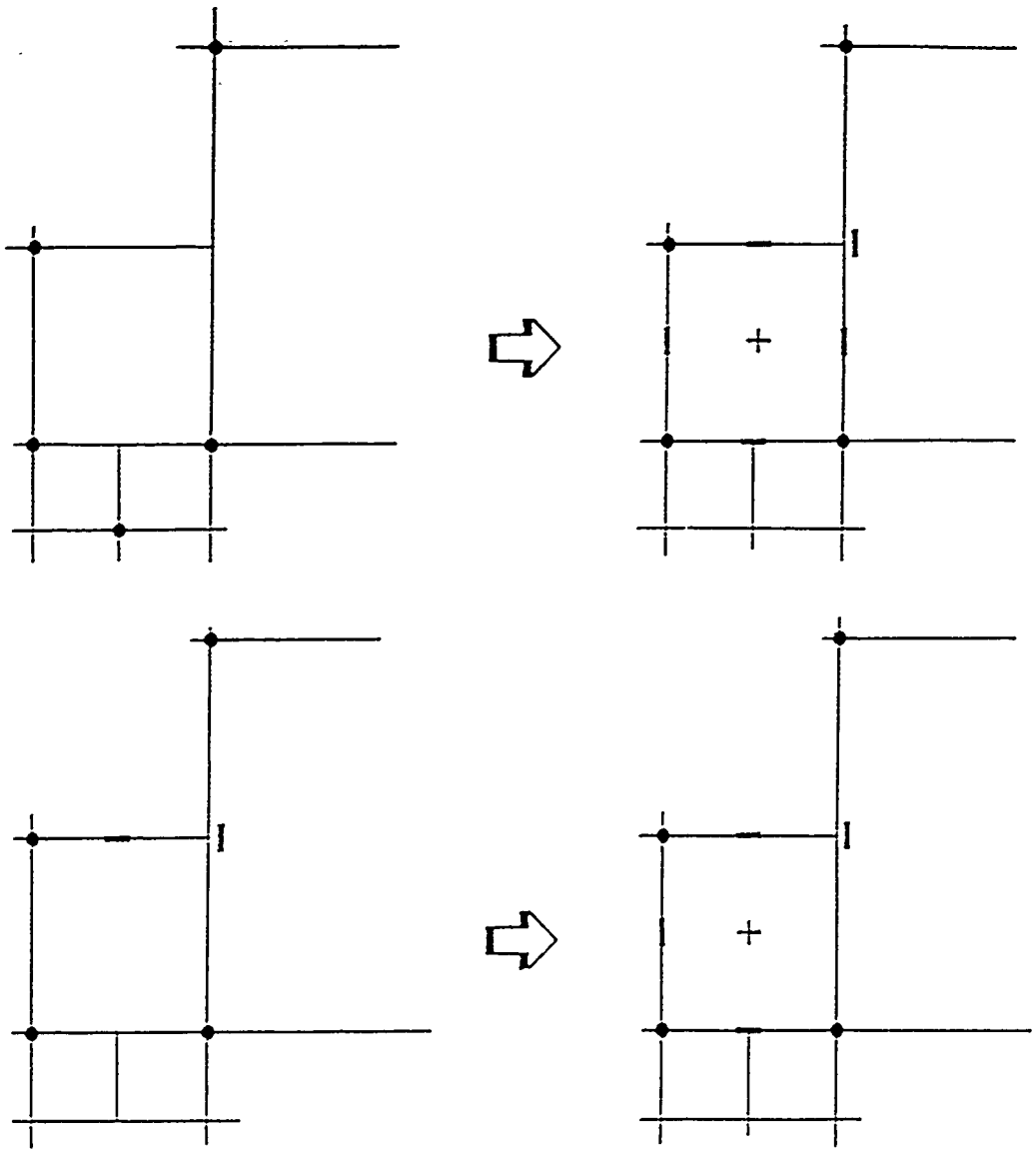


Figure A.2: p-Enrichments

A.2.2 *h*-Refinements

Only 1-irregular meshes are accepted in *h*-refinement. This is equivalent to saying that an element may not have more than two neighbors on a side. The rule is enforced by the following simple algorithm:

```
    • Set I=1, NELA(I) = NEL
10  • Determine up to eight neighbors of NELA(I) (at most two at a side)
      • FOR each side of NELA(I)
        • IF there is ony one neighbor on the side,
          say NEIG, THEN
          • determine neighbors of NEIG
          • IF NEIG has two neighbors on the side common
            with NELA(I) THEN
            • I=I+1, NELA(I) = NEIG
            • GO TO 10
          ENDIF
        ENDIF
      ENDFOR
    • Break element NELA(I) into four sons
    • I = I-1
    • IF (I.EQ.0) THEN STOP ELSE GO TO 10.
```

Typical situations of an element refinement are shown in Fig. A.3. When the element is of the first order, new nodes must be generated:

- on the element side, if the side is shared with two smaller neighbors,
- in the middle of the element.

Notice that on the sides shared with one element only due to enforced continuity, no new nodes are added and the local degrees of freedom become constrained. If a higher order node exists on a side of the element, the node is split into three nodes, one node of the first order and two of the same order as the split node. If the element itself is of a higher order its central node is split into 9 nodes:

- four central nodes
- four midpoints of new elements common edges and
- one node of the first order occupying the geometrical position of the original node.

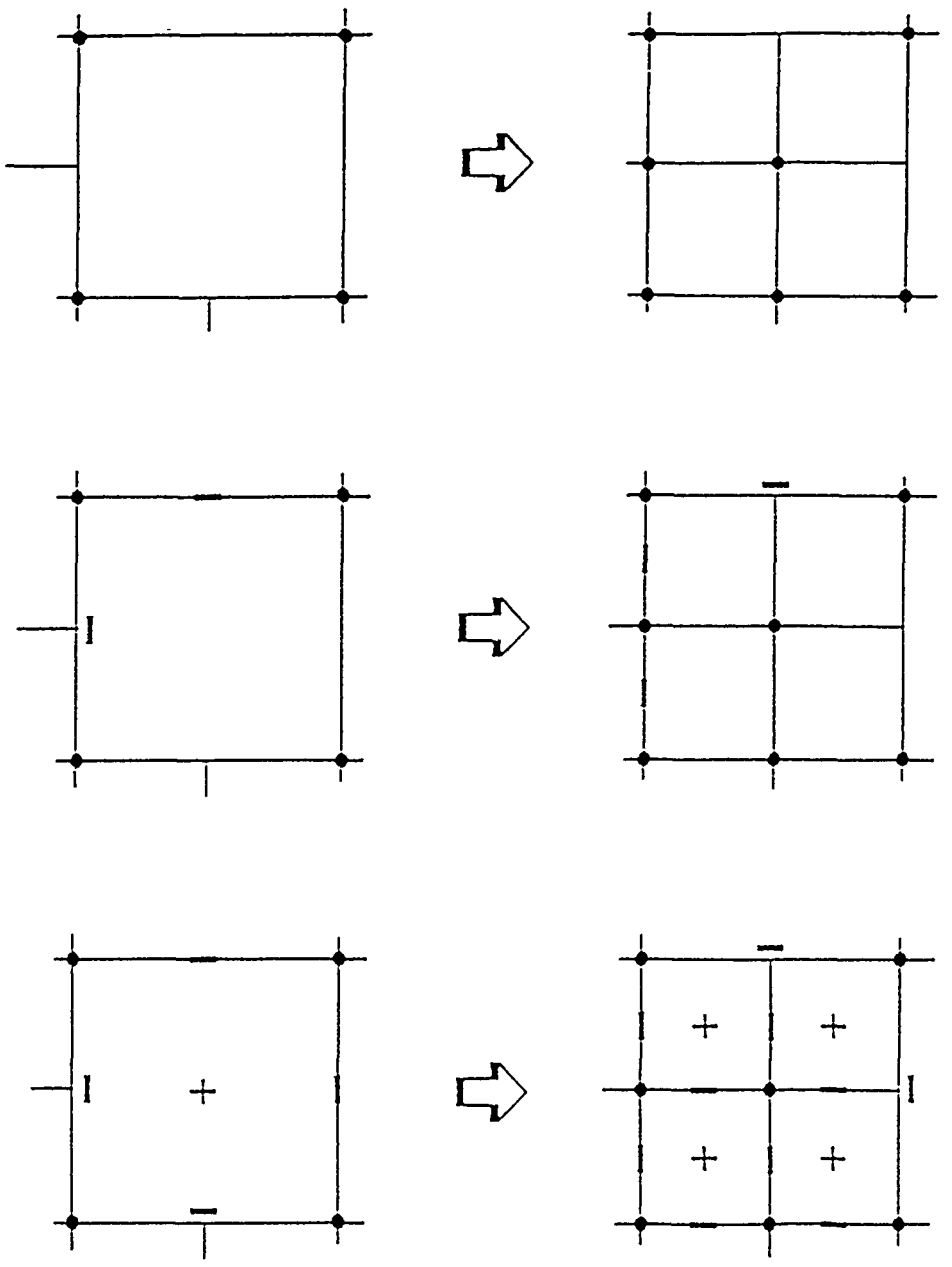


Figure A.3: h-Refinement

A.2.3 h-Unrefinements

The essential difference between the h-refinements and h-unrefinements is the conditionality of unrefinements. When required, the h-refinement of an element is always done and as a result of the enforced '1-irregularity rule' the element neighbors may be refined first. In order to *unrefine* a *previously* refined element the following conditions must be met:

1. the unrefined element may not have more than two neighbors on a side,
2. the four 'sons' central nodes and four boundary nodes, generated formerly, *must* be of the same order.

If two boundary nodes of higher order, belonging to the same side of the father element, have different order, the same (maximum) order is enforced.

A.3 A Linear Problem Solution

We consider an abstract linear problem in the form

$$\begin{cases} \text{Find } u_h \in X_h \text{ such that} \\ \\ a_h(u_h, v_h) = L_h(v_h) \quad \forall v_h \in X_h \end{cases} \tag{A.4}$$

where X_h is a FE space corresponding to the existing mesh and $a_h(\cdot,\cdot)$ and $L_h(\cdot)$ are FE approximations to the bilinear and linear forms defined in the Introduction.

The FE approximation is assumed to be continuous, i.e., X_h is a subspace of functions continuous on $\overline{\Omega}$.

A.3.1 Constrained Approximation

We assume that domain Ω can be represented as a union of quadrilateral or triangular finite elements $K_e, e = 1, \ldots, M$. More precisely,

$$\bar{\Omega} = \bigcup_{e=1}^{M} K_e \tag{A.5}$$

where

$$intK_e \cap intK_f = \emptyset \text{ for } e \neq f$$

Approximation on the element level.

Let K be a finite element with the corresponding space of shape functions

$$X_h(K).$$

The element degrees of freedom N_K, as usual, are viewed as linear functionals defined on $X_h(K)$. We assume that the set of degrees of freedom

$$\{\varphi_{i,K} : X_h(K) \to \mathbb{R} | i \in N_K\} \tag{A.6}$$

is $X_h(K)$-unisolvent. This in particular, implies that

1° $\varphi_{i,K}, i \in N_K$ are linearly independent

2° $\bigoplus_i \mathbb{R}\varphi_{i,K}, i \in N_K$ is a dual space to $X_h(K)$

The shape functions $\chi_{j,K}$ are defined as a dual basis to $\varphi_{i,K}$, i.e.,

$$< \varphi_{i,K}, \chi_{j,K} >= \delta_{ij} \qquad i,j \in N_K \tag{A.7}$$

and the FE approximation u_h within the element K is sought in the form

$$u_h = \sum_{i \in N_K} u_h^i \chi_{i,K}$$

where $u_h^i = \varphi_{i,K}(u_h)$.

In what follows we restrict ourselves to Lagrange and Hermite-type of degrees of freedom only. In other words, we assume that each of the functionals φ_i is of the form

$$\varphi \ : \ u \to D_x^k u(\xi_1, \ldots, \xi_k) \quad k = 0, 1, \ldots \tag{A.8}$$

where $D_x^k u$ denotes the k-th order differential of u evaluated at point x (as usual, $D_x^0 u = u(x)$). Vectors $\xi_1, \ldots, \xi_k$ at this point denote arbitrary vectors in $\mathbb{R}^2$. Thus, every degree of freedom can be identified with a point x and k vectors $\xi_1, \ldots, \xi_k$.

Construction of the unconstrained finite element space $\widetilde{X}_h$.

We introduce the following formal definition of the unconstrained finite element space $\widetilde{X}_h$:

$$\begin{aligned} \widetilde{X}_h \ = \ \big\{ & u_h : \Omega \to \mathbb{R} | u_h|_K \in X_h(K) \quad \forall x \\ & \text{such that: } \varphi_{K_e}(u_h/_{K_e}) = \varphi_{K_f}(u_h/_{K_f}) \\ & \text{for every two elements } K_e and K_f \end{aligned}$$

and degrees of freedom $\varphi_{K_e} : X_h(K_e) \to \mathbb{R}$, (A.9)
$\varphi_{K_f} : X_h(K_f) \to \mathbb{R}$ such that φ_{K_e} and φ_{K_f}
are defined through the same common point x
and vectors $\xi_1, \ldots, \xi_k\}$

(A.10)

Example. Consider a mesh of three rectangular Q^2 elements with standard Lagrange degrees of freedom. As shown in Fig. A.4. As one can see a function $u_h \in \widetilde{X}_h$ need not to be continuous. This is due to the presence of two irregular nodes A and B. At node A, for instance, associated with a degree of freedom for element K_2, from the point of view of element K_2 function u_h can take on an arbitrary value, while at the same point A , but treated as a point from element K_1, the value of u_h is determined uniquely by values at points C, D and E. In order to make the approximation continuous the value of u_h at A from the side of K_2 must be *forced* to be equal to the value of u_h at A from the side of K_1, which is equivalent to the elimination of the degree of freedom associated with point A by enforcing the constraint

$$u_h(A) = \alpha u_h(C) + \beta u_h(D) + \gamma u_h(E) \tag{A.11}$$

with proper coefficients α, β and γ.

Global degrees of freedom

Due to the construction of the space $\widetilde{X}_h$ we can introduce the global degrees of freedom identified by points (nodes) x and vectors $\xi_1, \ldots, \xi_k$. Formally, for every such point x and vectors $\xi_1, \ldots, \xi_k$ we define the linear functional Φ on $\widetilde{X}_h$

$$\Phi : \widetilde{X}_h \to \mathbb{R}, \Phi(u_h) = \varphi_K(u_h/_K), \tag{A.12}$$

where K is an element with the corresponding degree of freedom identified with point x and vectors $\xi_1, \ldots, \xi_k$. Note that due to the definition of $\widetilde{X}_h$ the global degrees of freedom are well defined.

The unconstrained base functions

The unconstrained base functions $\tilde{e}_i$ are introduced as a dual basis to the space of the global degrees of freedom, i.e.,

$$\langle \Phi_j, \tilde{e}_i \rangle = \delta_{ij} \tag{A.13}$$

Note that $\tilde{e}_i$ may be discontinuous.

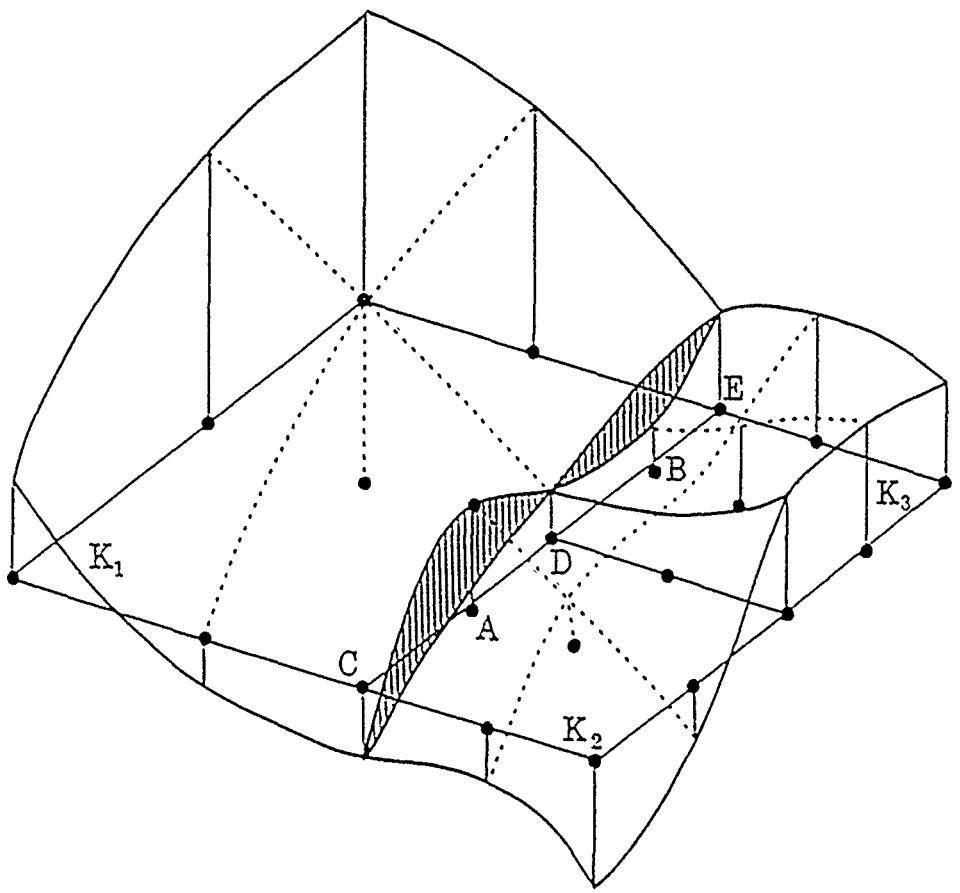


Figure A.4: Example of the unconstrained, discontinuous approximation.

Construction of the constrained finite element space X_h

At this moment, somewhat arbitrary, we classify all global degrees of freedom into two subsets:

- the set N^a of active degrees of freedom and
- the set N^c of constrained degrees of freedom

We assume that for each constrained degree of freedom Φ_i, $i \in N^c$, there exists a set $I(i)$ of corresponding active degrees of freedom, $I(i) \subset N^a$, and a vector $R_{ij}, j \in I(i)$ such that the following equality holds

$$\Phi_i(u_h) = \sum_{j \in I(i)} R_{ij} \Phi_j(u_h) \tag{A.14}$$

We introduce now the constrained finite element space X_h as

$$X_h = \left\{ u_h \in \widetilde{X}_h \;\; | \Phi_i(u_h) = \sum_{j \in I(i)} R_{ij} \Phi_j(u_h) \quad \forall\, i \in N^c \right\} \tag{A.15}$$

Assuming that the constraints are linearly independent we obtain the simple fact that X_h is dual to the space spanned by active degrees of freedom only. As usual, we define the base functions $e_j, j \in N^a$ as a dual basis to the set of active degrees of freedom:

$$e_j \in X_h \,, \;\; \langle \Phi_i, e_j \rangle = \delta_{ij} \;\; i, j \in N^a \tag{A.16}$$

Though, at this point the choice of constrained degrees of freedom is arbitrary, we implicitly assume that with the proper choice of constraints the resulting finite element space X_h consists of continuous functions only (compare the example).

Relation between unconstrained and constrained base f unctions

Let u_h be an arbitrary function from X_h. Then u_h must be of the following form

$$\begin{aligned} u_h &= \sum_{i \in N^a} u_i \tilde{e}_i + \sum_{j \in N^c} u_j \tilde{e}_j \\ &= \sum_{i \in N^a} u_i \tilde{e}_i + \sum_{j \in N^c} \sum_{k \in I(j)} R_{jk} u_k \tilde{e}_j \end{aligned}$$

Introducing for every $i \in N^a$ the set

$$S(i) = \{ j \in N^c | i \in I(j) \}$$

we rewrite u_h in the form

$$\begin{aligned} u_h &= \sum_{i \in N^a} u_i \tilde{e}_i + \sum_{k \in N^a} \sum_{j \in S(k)} u_k R_{jk} \tilde{e}_j \\ &= \sum_{i \in N^a} u_i \left(\tilde{e}_i + \sum_{j \in S(i)} R_{ji} \tilde{e}_j \right) \end{aligned} \tag{A.17}$$

We claim that functions

$$e_i = \tilde{e}_i + \sum_{j \in S(i)} R_{ji} \tilde{e}_j \qquad i \in N^a \tag{A.18}$$

form the dual basis to functionals $\Phi_i, i \in N^a$. Indeed

$$\langle \Phi_j, e_i \rangle = \langle \Phi_j, \tilde{e}_i \rangle + \sum_{k \in S(i)} R_{ki} \langle \Phi_j, \tilde{e}_k \rangle = \delta_{ij} \text{ since } S(i) \subset N^c$$

Calculation of the global load vector and stiffness matrix

Substituting (A.18) into both sides of (A.4) we get the formulas for the loaded vector and stiffness matrix.

$$L_h(e_i) = L_h(\tilde{e}_i) + \sum_{k \in S(i)} R_{ki} L_h(\tilde{e}_k) \tag{A.19}$$

$$\begin{aligned} a_h(e_i, e_j) &= a_h(\tilde{e}_i, \tilde{e}_j) \\ &+ \sum_{k \in S(i)} R_{ki} a_h(\tilde{e}_k, \tilde{e}_j) \\ &+ \sum_{l \in S(j)} R_{ij} a_h(\tilde{e}_i, \tilde{e}_l) \\ &+ \sum_{k \in S(i)} \sum_{l \in S(j)} R_{ki} R_{ij} a_h(\tilde{e}_k, \tilde{e}_i) \end{aligned} \tag{A.20}$$

A.3.2 Element Level Revisited – Modified Element Stiffness Matrix and Load Vector

Consider an element K. Let $N^a(K)$ and $N^c(K)$ denote active and constrained degrees of freedom. Assuming that, as usual, the load vector and stiffness matrix are calculated by summing up the contributions of all elements, i.e.,

$$L_h(u_h) = \sum_K L_{h,K}(u_h/_K) \tag{A.21}$$

and

$$a_h(u_h, v_h) = \sum_K a_{h,K}(u_h/_K, v_h/_K) \tag{A.22}$$

we arrive at the practical question, 'How does one calculate the contributions to the global load vector and stiffness matrix from element K?'

We introduce:

- the usual element load vector

$$\tilde{b}_{i,K} = L_{h,K}(\chi_{i,K}) \qquad i \in N^a \cup N^c \tag{A.23}$$

- and the element stiffness matrix

$$\tilde{a}_{ij,K} = a_{h,K}(\chi_{i,K}, \chi_{j,K}) \qquad i,j \in N^a \cup N^c \tag{A.24}$$

- the set of associated *active* degrees of freedom

$$N(K) = N^a(K) \cup \bigcup_{j \in N^c(K)} I(j) \tag{A.25}$$

  Notice that the two sets on the right-hand side of (A.25) need not be disjoint.

- the element contribution to the global load vector (modified element load vector)

$$b_{i,K} = L_{h,K}(e_i/_K) \qquad i \in N(K) \tag{A.26}$$

- the element contribution to the global stiffness matrix (modified element stiffness matrix)

$$a_{ij,K} = a_{h,K}(e_i/_K, e_j/_K) \qquad i,j \in N(K) \tag{A.27}$$

A.3.3 An Example

Definition of the hierarchical square Q^p master element

Setting $\hat{K} = [-1,1] \times [-1,1]$ we define the space of shape functions as

$$X_h(\hat{K}) = Q^p(\hat{K}) \tag{A.28}$$

where Q^P denotes the space of polynomials up to p-th order with respect to each of the variables separately.

The degrees of freedom are defined as follows: function values at four vertices:

$$u(-1,-1), u(1,-1), u(1,1), u(-1,1) \tag{A.29}$$

tangential derivatives (up to a multiplicative constant) up to p-th order associated with the midpoints of the four edges:

$$\lambda_k^{-1}\frac{\partial^k u}{\partial x^k}(0,-1) \quad k=2,\ldots,p$$

$$\lambda_k^{-1}\frac{\partial^k u}{\partial y^k}(1,0) \quad k=2,\ldots,p$$

$$\lambda_k^{-1}\frac{\partial^k u}{\partial x^k}(0,1) \quad k=2,\ldots,p$$

$$\lambda_k^{-1}\frac{\partial^k u}{\partial y^k}(-1,0) \quad k=2,\ldots,p$$

mixed order derivatives associated with the central node

$$\lambda_k^{-1}\lambda_l^{-1}\frac{\partial^{k+l} u}{\partial x^k \partial y^l}(0,0) \quad k,l=2,\ldots,p \tag{A.30}$$

One can easily see that the space of the shape functions $Q^p(\hat{K})$ is a tensor product of $P^p(-1,1)$ with itself and the degrees of freedom just introduced are simply the tensor products of the degrees of freedom for the 1-D element. More precisely, if $u \in Q^p$ then u is of the form

$$u(x,y) = \sum_k u^k v_k(x) w_k(y)$$

where $v_k, w_k \in P^p(-1,1)$, then each of the degrees of freedom can be represented in the form

$$\begin{aligned}(\varphi_i \otimes \varphi_j)(u) &= \sum_k u^k (\varphi_i \otimes \varphi_j)(v_k \otimes w_k) \\ &= \sum_k u^k \varphi_i(v_k)\cdot\varphi_j(w_k)\end{aligned}$$

This in particular implies that the corresponding shape functions can be identified with the *tensor products* of 1-D shape functions which are of the form

$$\chi_i(x)\chi_j(y) \quad i,j=0,1,\ldots,p \tag{A.31}$$

For $i,j=0,1$ we get the usual bilinear element with four nodal degrees of freedom.

Master element of an enriched order

Let $\widehat{K}$ be the element of the p-order. By adding additional shape functions corresponding to the $(p+1)$th order, together with the corresponding degrees of freedom, we get a well defined finite element whose space of shape functions includes more than Q^p. In particular, by adding all the additional degrees of freedom corresponding to the $(p+1)$th order, we pass to the element Q^{p+1} *without modifying* the existing shape functions and degrees of freedom. Equivalently, we can start with an element Q^{p+1} and eliminate some degrees of freedom passing to an uncomplete element of a lower order.

Subparametric hierarchical elements

Consider the master element of (possibly) an uncomplete order p. Even though p can be arbitrarily large, the element may be only Q^1-complete, which means that some of the nodes may be missing. An example of such an element is presented in Fig. A.6. An arbitrary location of the seven nodes in the plane (x, y) determine uniquely a map T from the master element into R^2, the components of T belonging to the uncomplete Q^2 space. More precisely, if $\hat{\varphi}_i, i = 1, \ldots, 9$ are the regular shape functions for the nine nodes biquadratic element, then

$$T(\hat{x}, \hat{y}) = \sum_{i=1}^{8} a_i \hat{\varphi}_i(\hat{x}, \hat{y}) \tag{A.32}$$

with the assumption that $a_5 = \frac{1}{2}(a_2 + a_3)$ and $a_7 = \frac{1}{2}(a_3 + a_4)$.

We have the classical definition of the subparametric element

$$K = T(\hat{K}) \tag{A.33}$$

with the space of test functions defined as

$$X_h(K) = \{u = \hat{u} \cdot T^{-1} | \hat{u} \in X_h(\hat{K})\} \tag{A.34}$$

and the degrees of freedom

$$< \varphi, u > = < \hat{\varphi}, \hat{u} > \text{ where } u = \hat{u} \circ T^{-1} \tag{A.35}$$

Interpretation of the degrees of freedom

The degrees of freedom associated with vertices are simply the function values evaluated at these points. The degrees of freedom associated with the midpoints of element edges and central nodes are more complicated. It follows from definition (A.34) that they may be

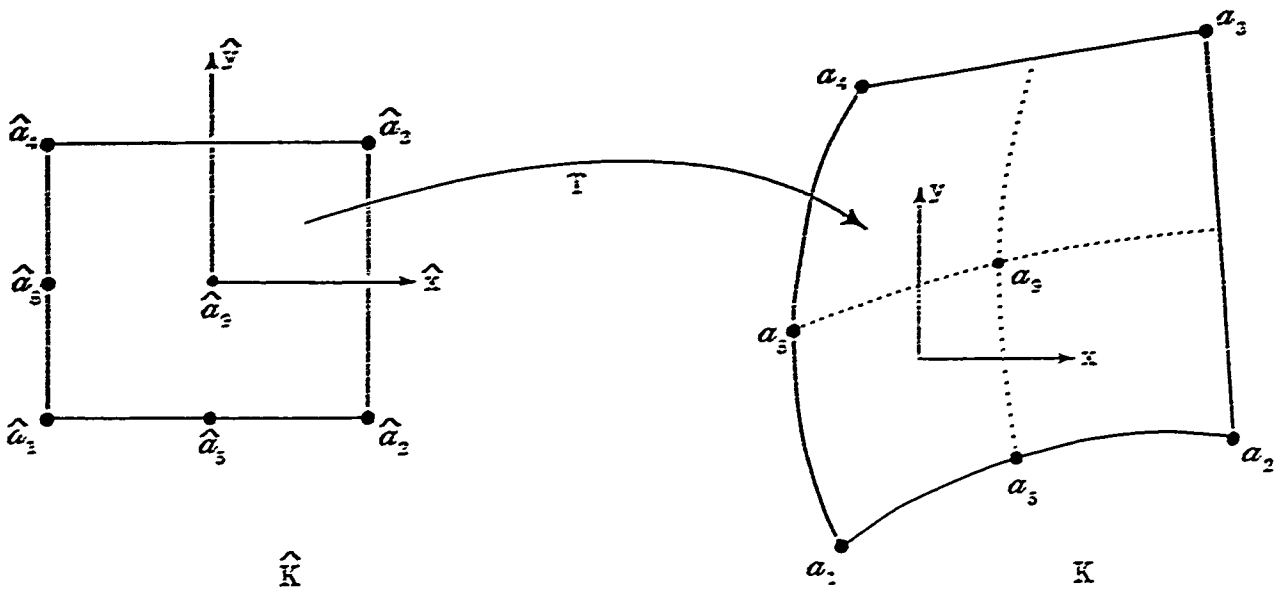


Figure A.5: Concept of the subparametric element.

interpreted as certain *linear combinations* of directional derivatives. The form of directional or mixed derivatives appropriate for the master element is preserved only if the map T is linear.

To be precise, all the definitions discussed in the previous section should be generalized to include the case of degrees of freedom defined as linear combinations of the Hermite degrees of freedom. We leave it for the reader as a simple exercise.

Constraints in the one dimensional case

We choose the scaling factors λ_p in (A.28) in such a way that the corresponding shape functions for the 1-D master element have the following form

$$\begin{aligned} \chi_0 &= \frac{1}{2}(1-\xi) \\ \chi_1 &= \frac{1}{2}(1+\xi) \\ \chi_p(\xi) &= \begin{cases} \xi^p - 1 & p = 2,4,6,\ldots \\ \xi^p - \xi & p = 3,5,7,\ldots \end{cases} \end{aligned} \tag{A.36}$$

In order to derive the explicit formula for the constraints coefficients R_{ij} (compare (A.14)), we consider three 1-dimensional elements: the master element (-1,1) and two elements (-1,0) and (1,0). Assume that all degrees of freedom for the 'big' element are active. The question is: what are the values the degrees of freedom for the small elements to be taken on so the functions spanned on the two small elements will exactly coincide with shape functions of the big element.

From the fact that (A.29) is a dual basis to (A.37) we get

$$\varphi_p(\chi_p) = \frac{1}{\lambda_p} p! - 1 \quad p = 2,3,\ldots \tag{A.37}$$

and therefore $\lambda_p = p!$

The transformation map from (-1,1) onto (-1,0) is of the form

$$x = -\frac{1}{2} + \frac{1}{2}\xi \tag{A.38}$$

with its inverse $\xi = 2x + 1$.

This yields the following formulas for the shape functions ${}^l\chi_p, p = 0,1,2,\ldots$ For the (left-hand side) element (-1,0) (compare definition (A.35)).

$${}^l\chi_0(X) = -x$$

$$
\begin{aligned}
{}^{l}\chi_1(X) &= x+1 \\
{}^{l}\chi_p(X) &= 1-(2x+1)^p \quad p=2,4,6,\ldots \\
{}^{l}\chi_p(X) &= (2x+1)^p-(2x+1) \quad p=3,5,7,\ldots
\end{aligned}
\tag{A.39}
$$

and the corresponding formulas for the degrees of freedom (compare (A.36))

$$
\begin{aligned}
<^{l}\varphi_0, u> &= u(-1) \\
<^{l}\varphi_1, u> &= u(0) \\
<^{l}\varphi_p, u> &= \frac{1}{2^p p!}\frac{d^p u}{dx^p}\left(-\frac{1}{2}\right)
\end{aligned}
\tag{A.40}
$$

Now let $u(x)(x = \xi$ for the master element) be any function spanned by the shape functions on (-1,1), i.e.

$$
u(x) = \sum_{q=0}^{k} \varphi_q(u)\chi_q(x) \tag{A.41}
$$

In order to represent $u(x)$ for $x \in (-1,0)$ in terms of the shape functions on (-1,0) we have to calculate the value of the degrees of freedom (A.41). We get

$$
\begin{aligned}
<^{l}\varphi_1, u> &= \varphi_0(u) <^{l}\varphi_0, \chi_0> + \sum_{q=1}^{k} \varphi_q(u) <^{l}\varphi_0, \chi_q) \\
&= <\varphi_0, u> \\
<^{l}\varphi_1, u> &= \varphi_0(u) <^{l}\varphi_1, \chi_0> + \varphi_1(u) <^{l}\varphi_1, \chi_1) \\
&+ \sum_{q=2}^{k} \varphi_q(u) <^{l}\varphi_1, \chi_q> \\
&= \frac{1}{2} <\varphi_0, u> + \frac{1}{2} <\varphi_1, u> + \sum_{q=2}^{k} {}^{l}R_{q1} <\varphi_q, u> \\
\text{where } {}^{l}R_{q1} &= <^{l}\varphi_1, \chi_q> \\
&= \begin{cases} 1 & \text{if q is even} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
\tag{A.42}
$$

and for $p \geq 2$

$$
\begin{aligned}
<^l \varphi_p, u> &= \varphi_0(u) <^l \varphi_p, \chi_0> + \sum_{q=1}^{k} \varphi_q(u) <^l \varphi_p, \chi_q> \\
&= 0 + \sum_{q=1}^{k} {}^lR_{qp} <\varphi_q, u>
\end{aligned}
$$

where

$$
\begin{aligned}
{}^lR_{qp} &= <^l \varphi_p, \chi_q> \\
&= \begin{cases} 0 & \text{for} \quad q < p \\ \dfrac{1}{2^q}\binom{q}{p} = \dfrac{(-1)^{p+q}}{2^q} \dfrac{q!}{p!(q-p)!} & \text{for } q \geq p \end{cases}
\end{aligned}
\tag{A.43}
$$

We apply the same procedure for the 'right-hand side' element (0,1) getting the following:

the transformation from $(-1,1)$ *onto* $(0,1)$

$$
x = \frac{1}{2} + \frac{1}{2}\xi \tag{A.44}
$$

with its inverse $\xi = 2x - 1$.

the shape functions ${}^r\chi_p, p = 0,1,2,\ldots$

$$
\begin{aligned}
{}^r\chi_0(x) &= 1 - x \\
{}^r\chi_1(x) &= x \\
{}^r\chi_p(x) &= \begin{cases} 1 - (2x-1)^p & p \text{ even} \\ (2x-1)^p - (2x-1) & p \text{ odd} \end{cases}
\end{aligned}
\tag{A.45}
$$

the degrees of freedom ${}^r\varphi_p$

$$
\begin{aligned}
<^r \varphi_0, u> &= u(0) \\
<^r \varphi_1, u> &= u(1) \\
<^r \varphi_p, u> &= \frac{1}{2^p p!} \frac{d^p u}{dx^p}\left(\frac{1}{2}\right)
\end{aligned}
\tag{A.46}
$$

the constraints

$$<^r \varphi_0, u>=\frac{1}{2}<\varphi_0, u>+\frac{1}{2}<\varphi_1, u>+\sum_{q=2}^{k} {}^rR_{q0}<\varphi_q, u>$$

where

$${}^rR_{q0} = \begin{cases} 1 & \text{if q is even} \\ 0 & \text{otherwise} \end{cases}$$

$$<^r \varphi_1, u>=<\varphi_1, u>$$

and for $p \geq 2$

$$<^r \varphi_p, u>=\sum_{q=2}^{k} {}^rR_{qp}<\varphi_q, u>$$

where

$$\begin{aligned} {}^rR_{qp} &= <^r \varphi_q, \chi_q> \\ &= \begin{cases} 0 & \text{for} \quad q<p \\ \frac{1}{2^q}\binom{q}{p} & \text{for} \quad q \geq p \end{cases} \end{aligned}$$

An example of the two arrays ${}^lR_{qp}$, ${}^rR_{qp}$, $q,p=0,\ldots,5$ is presented in Fig. A.7.

Constraints for the subparametric elements

Since the shape functions for the 2-D master element are defined as tensor products of the 1-D functions the results for the 1-D case hold exactly in the same form in the 2-D situation, with the only difference being that the calculated constraint equations have to be applied to the proper degrees of freedom (compare Fig. A.8). It follows from the definition of the subparametric elements that the constraints coefficients are *exactly the same* even when the elements have curved boundaries. This follows from the fact that the shape functions' behavior in a subparametric element on a part of its boundary depends exclusively upon the deformation of the part of the boundary, and therefore, any relation defined for the shape functions in the generic situation (on a master element) carries out immediately to the case of two 'small' elements sharing an edge with a 'big' element as long as the deformation of the edge is identical in all three elements. The situation is illustrated in Fig. A.8.

$$
{}^{l}R_{qp} = \begin{bmatrix}
1 & 1/2 & & & & & \\
 & 1/2 & & & & & \\
 & 1 & 1/4 & & & & \\
 & 0 & -3/8 & 1/8 & & & \\
 & 1 & 6/16 & -4/16 & 1/16 & & \\
 & 0 & -10/32 & 10/32 & -5/32 & 1/32 & \\
 & 1 & 15/64 & -20/64 & 15/64 & -6/64 & 1/64
\end{bmatrix}
$$

$$
{}^{r}R_{qp} = \begin{bmatrix}
1/2 & & & & & & \\
1/2 & 1 & & & & & \\
1 & & 1/4 & & & & \\
0 & & 3/8 & 1/8 & & & \\
1 & & 6/16 & 4/16 & 1/16 & & \\
0 & & 10/32 & 10/32 & 5/32 & 1/32 & \\
1 & & 15/64 & 20/64 & 15/64 & 6/64 & 1/64
\end{bmatrix}
$$

Figure A.6: The constraints coefficients for the sixth order of approximation. The unfilled coefficients are zero.

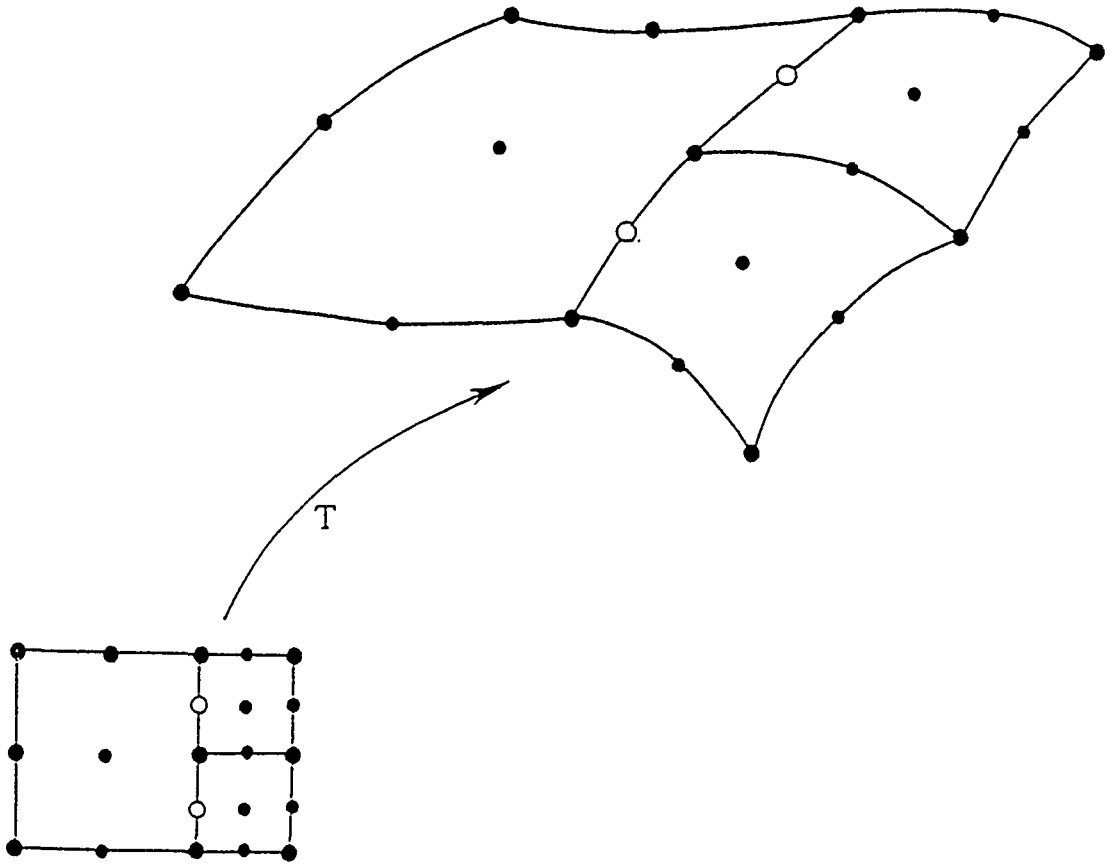


Figure A.7: Illustration of the constraints for the subparametric elements.

APPENDIX B

B Solution of Large Deformation Elasticity Problems by the *h*-*p* Finite Element Method

In order to demonstrate the techniques of artificial intelligence, we shall apply some of these concepts to the problems of elasticity involving large deformations. This class of problems was selected because its inherent nonlinearities allow solutions that depend upon the manner in which loads are applied. A conventional approach will be taken for these problems wherein the body to be studied is at first in an undeformed state called a reference configuration. Loads are then applied in a certain order and their magnitudes are increased until the desired loading configuration is obtained. The final deformed state of the body often depends upon the manner and rate in which these loads are introduced and so their aplication is a practical way to implement techniques of artificial intelligence. Namely, artificial intelligence methods will be used to select the loading manner so that an acceptable solution is obtained, preferably with a minimum of effort. Unacceptable configurations and/or solutions would be detected during the solution process and corrected using an intelligent decision making system.

B.1 Problem Formulation

Elasticity problems are typically formulated with respect to two possible coordinate systems: a referential system associated with the undeformed configuration of the body and a current system reflecting the position of a deformed body. Figure B.1 shows a body in the two systems and denotes quantities with respect to the reference configuration by capital letters and quantities with respect to the current configuration by lower case letters.

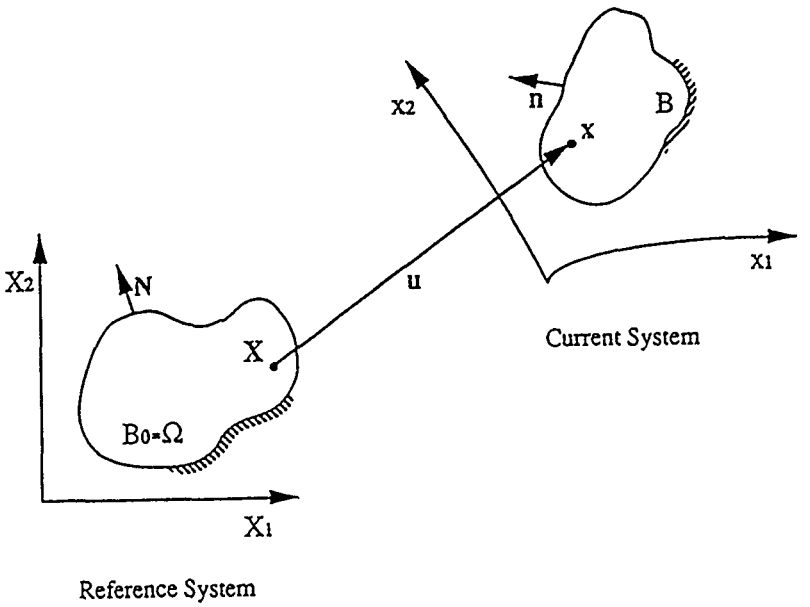


Figure B.1: Reference and current coordinate systems for a body experiencing deformation.

For simplicity, we shall always take the current coordinate sys: o coincide with the reference configuration (thus using a total Lagrangian description). ın order to reinforce the idea that all quantities are with respect to the reference system, capital letters will be used in almost all index notation.

Strain:

Since finite strains are admissible, Green's strain tensor will be used:

$$E_{IJ} = \frac{1}{2}\left(u_{I,J} + u_{J,I} + u_{K,I}u_{K,J}\right)$$

where the displacements u_I represent the changes in location of particles in the body.

$$u_I = x_I - X_I$$

Stress:

The second Piola-Kirchhoff stress tensor will be used in the stress/strain law according to:

$$T_{IJ} = T_{IJ}\left(E_{KL}\right)$$

Notice that this stress tensor is related to the Cauchy stress tensor by:

$$T_{IJ} = \det(\boldsymbol{F}) F_{Ii}^{-1} F_{Jj}^{-1} T_{ij}$$

where $\boldsymbol{F}$ is the deformation gradient

$$F_{iI} = x_{i,I}$$

and its inverse:

$$F_{Ii}^{-1} = X_{I,i}$$

Equilibrium equations:

In Lagrangian coordinates, the equations of equilibrium can be written as:

$$S_{IJ,J} + f_I = 0$$

where S_{IJ} is the first Piola-Kirchhoff stress tensor and f_I is the body force referred to the area in the reference configuration.

In terms of the second Piola-Kirchhoff tensor, these equations become:

$$(T_{IJ}F_{JK})_{,I} + f_K = 0$$

where, by virtue of the choice of coordinate systems:

$$F_{JK} = F_{jK} = x_{j,K} = \delta_{JK} + u_{J,K}$$

Thus, in terms of stress and displacement the equilibrium equations can be written:

$$[(\delta_{IK} + u_{I,K})\, T_{KJ}]_{,J} + f_I = 0$$

Notice that even if a linear relationship between stress and strain is used, the equations are still nonlinear with respect to the displacements due to the mixed derivative terms in the strain tensor.

Boundary conditions:

Suitable boundary conditions for this problem might include prescribed displacements (with respect to the reference configuration) on a portion Γ_D of the domain boundary

$$u_I = \hat{u}_I \text{ on } \Gamma_D$$

and prescribed tractions on the remaining portion Γ_T of the boundary

$$S_{IJ}N_J = t_I \text{ on } \Gamma_t$$

or:

$$T_{IJ}\,(\delta_{JK} + u_{J,K})\, N_K = t_I$$

where N_J is the normal to the boundary on the reference configuration and t_I is a prescribed surface traction also with respect to the reference configuration.

B.2 Variational Formulation

Multiplying the equilibrium equations by a test function, integrating the result over the problem domain (in the reference configuration) and applying Green's identity, we arrive at the variational formulation

$$\int_\Omega T_{IJ}\delta E_{IJ}d\Omega = \int_\Omega f_I v_I d\Omega + \int_{\Gamma_T} t_I v_I d\Gamma$$

where the variation of E_{IJ} on v is denoted δE_{IJ} where

$$\delta E_{IJ} = (\delta_{IK} + u_{I,K})\, v_{K,J}$$

The above formulation corresponds directly to the principle of virtual work (merely replace the test function v_I by a virtual displacement δu_I). Furthermore, this formulation is nonlinear since T_{IJ} is a nonlinear function of u and in general, the body force f_I and surface tractions t_I may depend upon u.

At this point, our problem may be stated in terms of the variational formulation according to:

Find $\boldsymbol{u} \in \boldsymbol{H}$ such that the variational formulation is satisfied for all $\boldsymbol{v} \in \boldsymbol{V}$.

In this case, our space of trial functions consists of

$$\boldsymbol{H} = \left\{ \boldsymbol{w} \in \left[H^1(\Omega)\right]^N \;:\; \boldsymbol{w} = \hat{\boldsymbol{u}} \text{ on } \Gamma_D \right\}$$

and the space of test functions

$$\boldsymbol{V} = \left\{ \boldsymbol{v} \in \left[H^1(\Omega)\right]^N \;:\; \boldsymbol{v} = 0 \text{ on } \Gamma_D \right\}$$

where in both spaces N is the dimension of the problem.

B.3 Approximate Problem

The spaces of the variational problem can be restricted to finite element subspaces of $\boldsymbol{H}$ and $\boldsymbol{V}$ according to:

$$u_I = \sum_{\alpha=1}^{M} u_{I\alpha}\psi_\alpha \in H^h \subset H$$

$$v_I = \sum_{\alpha=1}^{M} v_{I\alpha}\psi_\alpha \in V^h \subset V$$

where $u_{I\alpha}$ is the degree-of-freedom for the I-th component of $\boldsymbol{u}$ associated with the shape function ψ_α and M is the total number of shape functions per component.

Substituting these expressions for the finite element trial and test functions into the variational formulation, we obtain:

$$\begin{aligned}\int_\Omega T_{IJ} B_{IJKL} v_{K\alpha} \psi_{\alpha,L} d\Omega &= \int_\Omega f_I v_{I\alpha} \psi_\alpha d\Omega \\ &+ \int_{\Gamma_T} t_I v_{I\alpha} \psi_\alpha d\Gamma\end{aligned}$$

where $B_{IJKL} = (\delta_{IK} + u_{I\beta}\psi_{\beta,K})\delta_{JL}$ and represents the nonlinear geometric matrix:

$$B_{IJKL} = \frac{\partial E_{IJ}}{\partial u_{K,L}}$$

This formulation can be written as the set of equations

$$\begin{aligned}\int_\Omega T_{KJ} B_{KJIL} \psi_{\alpha,L} d\Omega &= \int_\Omega f_I \psi_\alpha d\Omega \\ &+ \int_{\Gamma_T} t_I \psi_\alpha d\Omega\end{aligned}$$

where we have an equation for each combination of component I and shape function α.

B.4 Solution by Newton's Method

Although a varitional form of the finite element problem has been obtained, it must be linearized before a solution can be computed. In order to accomplish this, a simple iterative technique based upon Newton's method shall be employed.

First we write the system of nonlinear equations as the functional

$$I_{I\alpha} = I_{I\alpha}(u_{J\beta}) = L_{I\alpha} - R_{I\alpha} = 0$$

where $L_{I\alpha}$ and $R_{I\alpha}$ are the left and right hand sides of our nonlinear equations:

$$L_{I\alpha} = \int_\Omega T_{KJ} B_{KJIL} \psi_{\alpha,L} d\Omega$$

and

$$R_{I\alpha} = \int_\Omega f_I \psi_\alpha d\Omega + \int_{\Gamma_T} t_I \psi_\alpha d\Gamma$$

Thus our problem may be stated:

Find a set of degrees of freedom, $u_{J\beta}$ such that

$$I_{I\alpha}(u_{J\beta}) = 0$$

for every combination of I and α.

Expanding $I_{I\alpha}$ in a Taylor series about a particular guess $u^k_{J\beta}$

$$I_{I\alpha}\left(u^k_{J\beta}+\Delta u^k_{J\beta}\right)=I_{I\alpha}\left(u^k_{J\beta}\right)+\Delta u^k_{J\beta}\frac{\partial I_{I\alpha}\left(u^k_{J\beta}\right)}{\partial u_{J\beta}}+\ldots$$

Notice that if the loads associated with the problem are changing in time that $I_{I\alpha}$ is also a function of time. We assume, however, that all time-dependent loads are known or at least that time is unimportant since we are concerned only with a final steady-state solution. Namely, this formulation is quasistatic in the sense that at an indicated moment in time, the values of any time-dependent loads are known and a steady-state solution is sought.

Truncating the Taylor series after the first term and setting the left hand side equal to zero, we obtain the iterative scheme

$$K^k_{I\alpha J\beta}\Delta u^k_{J\beta}=-I_{I\alpha}\left(u^k_{J\beta}\right)$$

$$u^{k+1}_{J\beta}=u^k_{J\beta}+\Delta u^k_{J\beta}$$

where $K^k_{I\alpha J\beta}$ is the tangent stiffness matrix defined according to:

$$K^k_{I\alpha J\beta}=\frac{\partial I_{I\alpha}\left(u^k_{J\beta}\right)}{\partial u_{J\beta}}$$

For the case where *only* $L_{I\alpha}$ is a function of $u_{J\beta}$, the tangent stiffness matrix may be written:

$$\begin{aligned}K_{I\alpha J\beta} &= \int_\Omega \frac{\partial}{\partial u_{J\beta}}\left(T_{MN}B_{MNIL}\right)\psi_{\alpha,L}d\Omega\\ &= \int_\Omega \frac{\partial}{\partial u_{J,S}}\left(T_{MN}B_{MNIL}\right)\psi_{\beta,S}\psi_{\alpha,L}d\Omega\end{aligned}$$

since

$$\frac{\partial(\cdot)}{\partial u_{J\beta}}=\frac{\partial(\cdot)}{\partial u_{J,S}}\frac{\partial u_{J,S}}{\partial u_{J\beta}}=\frac{\partial(\cdot)}{\partial u_{J,S}}\psi_{\beta,S}$$

Expanding the derivative:

$$\begin{aligned}K_{I\alpha J\beta} &= \int_\Omega C_{MNPQ}B_{PQJS}B_{MNIL}\psi_{\beta,S}\psi_{\alpha,L}d\Omega\\ &+ \int_\Omega T_{MN}G_{MNILJS}\psi_{\beta,S}\psi_{\alpha,L}d\Omega.\end{aligned}$$

where:

$$C_{MNPQ}=\frac{\partial T_{MN}}{\partial E_{PQ}}$$

is usually a tensor of material dependent constants and

$$G_{MNILJS} = \frac{\partial B_{MNIL}}{\partial u_{JS}}$$

simplifies to:

$$G_{MNILJS} = \delta_{NL}\delta_{JM}\delta_{SI}$$

Using this scheme, iterations are performed until

$$Del^{k}_{J\beta} < \varepsilon_{Del}$$

and

$$I_{I\alpha}\left(u^{k}_{J\beta}\right) < \varepsilon_{I}$$

The converged iterate can then be used as the first guess for the new loading configuration at the next moment in time. This process is then repeated until a converged solution is obtained for the final loading configuration. A flow chart in Fig. B.2 summarizes the solution process.

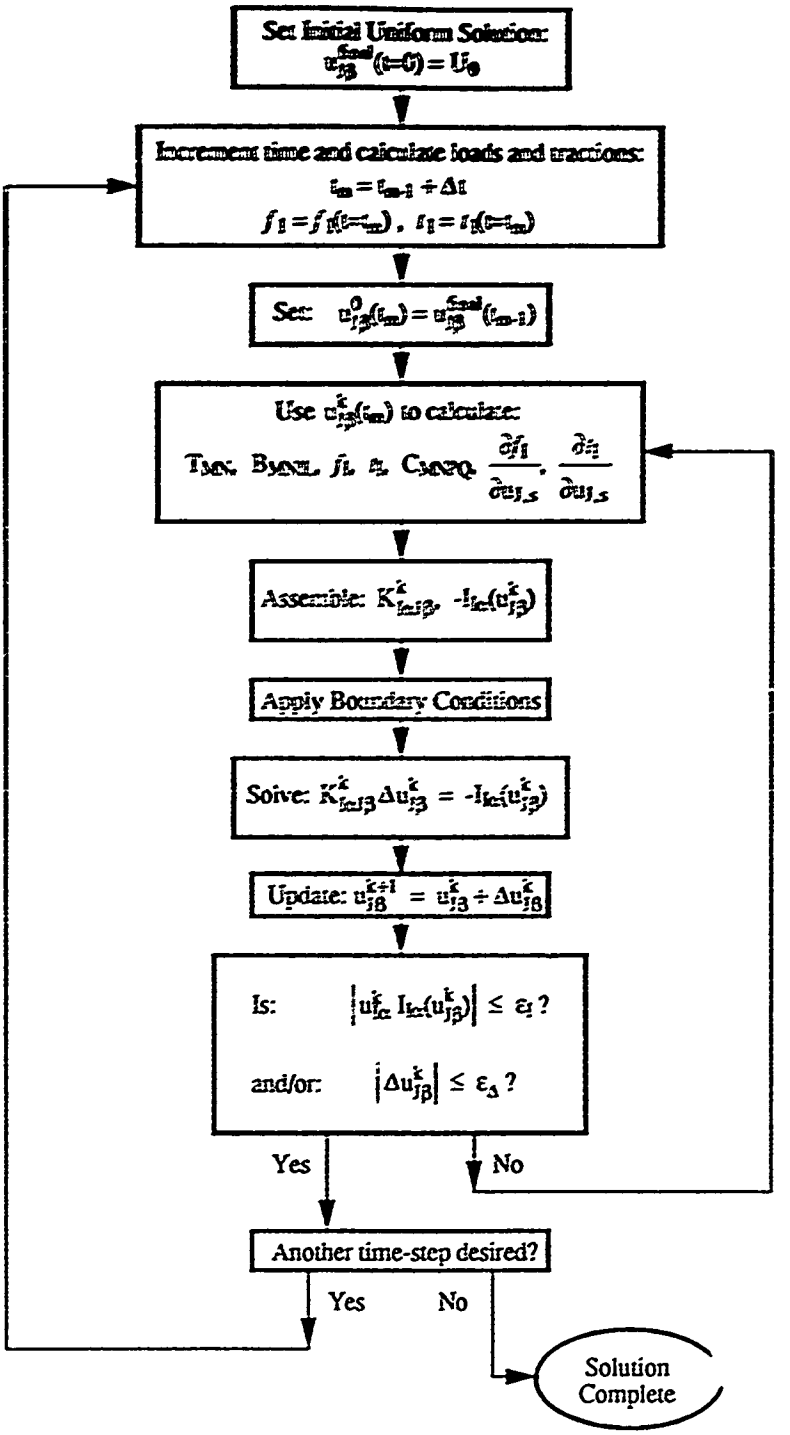


Figure B.2: Solution process of large deformation elasticity problems by Newton's method.

APPENDIX C

C Listings of Knowledge Bases for Coupled PHLEX–NEXPERT Environment

This appendix summarizes listings of knowledge bases developed in the project for an automated PHLEX-NEXPERT environment, discussed in Section 8. These listings follow the actual structure of NEXPERT's knowledge base files. For simplicity, some of the items not necessary for the presentation of the basic concepts have been left out.

Each listing presents a definition of classes and objects (including name and slots), metaslots associated with selected slots and rules. Each rule has a left hand side in the form of conditions, a boolean hypothesis and a right hand side, which defines additional actions associated with this rule. A more detailed explanation of these concepts can be found in the NEXPERT OBJECT manual (proprietary document of Neuron Data, Inc.).

C.1 Strategy Selection Knowledge Base

This simple expert system selects the type of solver for the finite element analysis. The solvers presently available are frontal and iterative. The list of objects and rules is presented below.

Objects

```
(OBJECT=        Code
        (PROPERTIES=
                frontal_solver
                iterative_solver
                ndofs
                ordering
        )
)

(OBJECT=        Itslv
        (PROPERTIES=
                accel
                eps
                freq
                ifgs
                maxit
                print
        )
)

(OBJECT=        solver
        (PROPERTIES=
                ndof_max_frnt
                ndof_opt_frnt
        )
```

Meta–slots

```
(SLOT= Code.ordering
        (INITVAL=       0)
)

(SLOT= solver.ndof_max_fmt
        (SOURCES=
                (Retrieve               ("kbases/solver.nxp"))
        )
)

(SLOT= solver.ndof_opt_fmt
        (SOURCES=
                (Retrieve               ("kbases/solver.nxp"))
        )
)
```

Rules

```
(RULE= front_small_nel
        (LHS=
                (<      (Code.ndofs-solver.ndof_opt_fmt)                (0))
        )
        (HYPO= Code.frontal_solver)
        (RHS=
                (Let    (Code.iterative_solver)         (FALSE))
        )
)

(RULE= front_good_ord
        (LHS=
                (<      (Code.ndofs-solver.ndof_max_fmt)                (0))
                (<=     (Code.ordering)     (2))
        )
        (HYPO= Code.frontal_solver)
        (RHS=
                (Let    (Code.iterative_solver)         (FALSE))
        )
)

(RULE= Iterative
        (LHS=
                (No     (Code.frontal_solver))
        )
        (HYPO= Code.iterative_solver)
        (RHS=
                (Retrieve               ("kbases/solver.nxp")
        (FIELDS="Itslv.accel","Itslv.eps","Itslv.freq",\
         "Itslv.ifgs","Itslv.maxit","Itslv.print")
        )
)
```

C.2 Performance Control Knowledge Base

This expert system monitors the performance of computations, in particular of Newton–type iterative procedures for nonlinear problems. A general structure of this system is presented in Section 8.3.2.

Objects

```
(OBJECT=        Code
        (PROPERTIES=
                deltat
                deltnew
                error
                ertnew
                ertol
                miter
                mitnew
                niter
        )
)

(OBJECT=        User
        (PROPERTIES=
                runtyp
        )
)

(OBJECT=        Nonl
        (PROPERTIES=
                converged
                converges
                convtype
                decided
                decision
                div_solprob
                diverges
                ertolmi
                ertolmx
                fixedertol
                mbacked
                mitadd
                mitmax
                nbacked
                need_back_off
                optniter
                reset
                reseted
                stagnates
        )
)

(OBJECT=        Solver
        (PROPERTIES=
                problem
        )
)
```

```
(SLOT= User.runtyp
       PROMPT="Specify type of control of nonlinear problem:";
       (SOURCES=
              (AskQuestion      (User.runtyp)      (NOTKNOWN))
       )
)

(SLOT= Nonl.ertolmi
       (SOURCES=
              (Retrieve         ("kbases/nonldat.nxp")
              (RunTimeValue     (1.0e-06))
       )
)

(SLOT= Nonl.ertolmx
       (SOURCES=
              (Retrieve         ("kbases/nonldat.nxp")
              (RunTimeValue     (0.01))
       )
)

(SLOT= Nonl.mbacked
       (SOURCES=
              (Retrieve         ("kbases/nonldat.nxp")
              (RunTimeValue     (4))
       )
)

(SLOT= Nonl.mitadd
       (SOURCES=
              (Retrieve         ("kbases/nonldat.nxp")
              (RunTimeValue     (3))
       )
)

(SLOT= Nonl.mitmax
       (SOURCES=
              (Retrieve         ("kbases/nonldat.nxp")
              (RunTimeValue     (20))
       )
)

(SLOT= Nonl.nbacked
       (INITVAL=        0)
)

(SLOT= Nonl.optniter
       (SOURCES=
              (Retrieve         ("kbases/nonldat.nxp")
              (RunTimeValue     (6))
       )
)
```

```
(SLOT= Nonl.reset
        (CACTIONS=
                (Reset   (Nonl.decided))
                (Reset   (Nonl.fixedertol))
                (Reset   (Nonl.div_solprob))
                (Reset   (Nonl.reseted))
                (Do      (Code.miter)      (Code.mitnew))
                (Do      (Code.deltat)     (Code.deltnew))
                (Do      (Code.ertol)      (Code.ertnew))
        )
)

(SLOT= Solver.problem
        (SOURCES=
                (RunTimeValue    (NOTKNOWN))
        )
)
```

Rules

```
(RULE= Nonl_reseted
        (LHS=
                (Yes     (Nonl.reset))
        )
        (HYPO= Nonl.reseted)
        (RHS=
                (Reset   (Nonl.reset))
        )
)


(RULE= Nonl_converges
        (LHS=
                (Is      (Nonl.convtype)   ("converges"))
        )
        (HYPO= Nonl.converges)
)

(RULE= Too_many_back
        (LHS=
                (Yes     (Nonl.need_back_off))
                (>       (Nonl.nbacked-Nonl.mbacked)        (0))
        )
        (HYPO= Nonl.decided)
        (RHS=
                (Let     (Nonl.decision)   ("terminate"))
                (Show    ("kbases/stopnonl1.txt")
        )
)
```

```
(RULE= Nonl_converged
        (LHS=
                (Yes    (Nonl.converged))
        )
        (HYPO= Nonl.decided)
        (RHS=
                (Let    (Nonl.decision)    ("conttime"))
                (Do     (Code.deltat*Nonl.optniter/Code.niter)          (Code.deltnew))
                (Strategy           (@EXHBWRD=FALSE;))
                (Do     (0)     (Nonl.nbacked))
        )
)

(RULE= Nonl_back_off
        (LHS=
                (Yes    (Nonl.need_back_off))
                (<=     (Nonl.nbacked-Nonl.mbacked)         (0))
        )
        (HYPO= Nonl.decided)
        (RHS=
                (Let    (Nonl.decision)    ("back_off"))
                (Do     (Nonl.nbacked+1)            (Nonl.nbacked))
        )
)

(RULE= No_analysis
        INFCAT=20;
        (LHS=
                (IsNot  (User.runtyp)      ("automatic"))
                (Is     (User.runtyp)      ("algorithm"))
        )
        (HYPO= Nonl.decided)
        (RHS=
                (Let    (Nonl.decision)    ("I_refuse"))
                (Strategy           (EXHBWRD=FALSE;))
        )
)

(RULE= Extend_miter
        (LHS=
                (Yes    (Nonl.converges))
                (No     (Nonl.converged))
                (<=     (Code.miter+Nonl.mitadd-Nonl.mitmax)        (0))
        )
        (HYPO= Nonl.decided)
        (RHS=
                (Let    (Nonl.decision)    ("contiter"))
                (Do     (Code.miter+Nonl.mitadd)            (Code.mitnew))
        )
)

(RULE= Nonl_div_due_to_solver
        (LHS=
                (Yes    (Nonl.diverges))
                (Yes    (Solver.problem))
        )
        (HYPO= Nonl.div_solprob)
        (RHS=
                (Show   ("kbases/nonldivsol.txt")
        )
)
```

```
(RULE= Nonl_diverges
        (LHS=
                (Is      (Nonl.convtype)    ("diverges"))
        )
        (HYPO= Nonl.diverges)
)

(RULE= Check_ertolmi
        (LHS=
                (Yes     (Nonl.converges))
                (Yes     (Nonl.stagnates))
                (<       (Code.ertol-Nonl.ertolmi)          (0))
                (<       (Code.error-Nonl.ertolmi)          (0))
        )
        (HYPO= Nonl.fixedertol)
        (RHS=
                (Show    ("kbases/fixertolmi.txt")
                (Do      (Nonl.ertolmi)     (Code.ertnew))
        )
)
(RULE= back_off_stagn
        (LHS=
                (Yes     (Nonl.stagnates))
                (No      (Nonl.converged))
        )
        (HYPO= Nonl.need_back_off)
        (RHS=
                (Do      (Code.deltat/2.0)          (Code.deltnew))
        )
)

(RULE= back_off_div
        (LHS=
                (Yes     (Nonl.diverges))
                (No      (Nonl.converged))
        )
        (HYPO= Nonl.need_back_off)
        (RHS=
                (Do      (Code.deltat/2.0)          (Code.deltnew))
        )
)

(RULE= Nonl_stagnates
        (LHS=
                (Is      (Nonl.convtype)    ("stagnates"))
        )
        (HYPO= Nonl.stagnates)
)

(RULE= Change_convergence
        (LHS=
                (Yes     (Nonl.converges))
                (No      (Nonl.converged))
                (>       (Code.miter+Nonl.mitadd-Nonl.mitmax)       (0))
        )
        (HYPO= Nonl.stagnates)
)
```

C.3 Result Verification Knowledge Base

This expert system performs the verification of the final results, in particular the correctness of the mathematical model, the satisfaction of basic design criteria, etc. A general structure of this system is presented in Section 8.6.

Classes and Objects

```
(CLASS=solids
        (PROPERTIES=
                ar_eps90
                ar_str90
                dimmax
                dimmin
                dispmax
                dispmin
                disrmax
                epsmax
                epsmaxabs
                finrotmax
                finrotmin
                large_rigid_rot
                large_rot
                mattype
                rotavg
                rotmax
                rotmin
                rotrigid
                small_displ
                small_rot
                str_eff
                strlimel
                strmax
                strmaxabs
                strmin
                theory_disp
                theory_mat
                theory_strain
        )
)

(OBJECT=        solid_generator
        (PROPERTIES=
                n
                new_solid
        )
)

(OBJECT=        Versolid
        (PROPERTIES=
                areasmall
                disrbig
                disrsmall
                epssmall
                rotdifsml
                rotsmall
        )
)
```

```
(SLOT= solid_generator.n
       (INITVAL=        1)
)

(SLOT= Versolid.areasmall
       (SOURCES=
              (Retrieve          ("kbases/verdat.nxp")
              (RunTimeValue      (0.05))
       )
)

(SLOT= Versolid.disrbig
       (SOURCES=
              (Retrieve          ("kbases/verdat.nxp")
              (RunTimeValue      (5.0))
       )
)

(SLOT= Versolid.disrsmall
       (SOURCES=
              (Retrieve          ("kbases/verdat.nxp")
              (RunTimeValue      (0.05))
       )
)

(SLOT= Versolid.epssmall
       (SOURCES=
              (Retrieve          ("kbases/verdat.nxp")
              (RunTimeValue      (0.05))
       )
)

(SLOT= Versolid.rotdifsml
       (SOURCES=
              (Retrieve          ("kbases/verdat.nxp")
              (RunTimeValue      (3.0))
       )
)

(SLOT= Versolid.rotsmall
       (SOURCES=
              (Retrieve          ("kbases/verdat.nxp")
              (RunTimeValue      (6.0))
       )
)
```

Rules

```
(RULE= excess_displ
        (LHS=
                (>=     ({lsolidsl}.disrmax-Versolid.disrbig)           (0.0))
        )
        (HYPO= excessive_displ)
)

(RULE= large_displ
        (LHS=
                (>      (<lsolidsl>.disrmax-Versolid.disrsmall)         (0.0))
        )
        (HYPO= large_displ)
)

(RULE= large_rigid_rot
        (LHS=
                (Yes    (large_rot_exist))
                (Yes    (<lsolidsl>.large_rot))
                (>      (<lsolidsl>.rotrigid-Versolid.rotsmall)         (0.0))
                (<      (ABS(<lsolidsl>.rotmax-<lsolidsl>.rotmin)-Versolid.rotdifsml)           (0.0))
        )
        (HYPO= large_rigid_rot_exist)
        (RHS=
                (Let    (<lsolidsl>.large_rigid_rot)            (TRUE))
        )
)

(RULE= large_rot
        (LHS=
                (>      (<lsolidsl>.rotmax-Versolid.rotsmall)           (0.0))
        )
        (HYPO= large_rot_exist)
        (RHS=
                (Let    (<lsolidsl>.large_rot)          (TRUE))
        )
)

(RULE= mooney_resolved
        (LHS=
                (Is     (<lsolidsl>.mattype)            ("MonRiv"))
        )
        (HYPO= mattype_resolved)
        (RHS=
                (Let    (<lsolidsl>.theory_mat)         ("elastic"))
                (Let    (<lsolidsl>.theory_strain)              ("large_stn"))
        )
)
```

```
(RULE= hook_resolved
        INFCAT=10;
        (LHS=
                (Is     (<lsolidsl>.mattype)        ("Hooke"))
        )
        (HYPO= mattype_resolved)
        (RHS=
                (Let    (<lsolidsl>.theory_mat)     ("elastic"))
                (Let    (<lsolidsl>.theory_strain)          ("small_strn"))
        )
)

(RULE= may_large_strains
        (LHS=
                (>      (<lsolidsl>.epsmaxabs-Versolid.epssmall)                (0.0))
                (<=     (<lsolidsl>.epsmaxabs*0.9-Versolid.epssmall)            (0.0))
                (IsNot  (<lsolidsl>.theory_strain)          ("large_strn"))
                (<=     (<lsolidsl>.ar_eps90-Versolid.areasmall)                (0.0))
        )
        (HYPO= may_large_strains)
)

(RULE= may_small_displ
        (LHS=
                (IsNot  (<lsolidsl>.theory_disp)            ("small_disp"))
                (<      ({lsolidsl}.epsmaxabs-Versolid.epssmall)                (0.0))
                (<      ({lsolidsl}.rotmax-Versolid.rotsmall)       (0.0))
                (<      ({lsolidsl}.disrmax-Versolid.disrsmall)     (0.0))
        )
        (HYPO= may_small_disp)
        (RHS=
                (Show   ("kbases/may_smdis.txt")
        )
)

(RULE= need_large_disp2
        (LHS=
                (>      (<lsolidsl>.epsmaxabs*0.9-Versolid.epssmall)            (0.0))
                (IsNot  (<lsolidsl>.theory_disp)            ("large_disp"))
        )
        (HYPO= need_large_disp)
)

(RULE= need_large_disp1
        (LHS=
                (>      (<lsolidsl>.epsmaxabs-Versolid.epssmall)                (0.0))
                (IsNot  (<lsolidsl>.theory_disp)            ("large_disp"))
                (>      (<lsolidsl>.ar_eps90-Versolid.areasmall)                (0.0))
        )
        (HYPO= need_large_disp)
)
```

```
(RULE= need_large_disp
        (LHS=
                (Yes    (large_not_exist))
                (Yes    (<solidsb>.large_not))
                (>      (ABS(<solidsb>.romax-<solidsb>.romin)-Versolid.rodiffsmall)      (0.0))
                (IsNot  (<solidsb>.theory_disp)          ("large_disp"))
        )
        (HYPO= need_large_disp)
)

(RULE= need_large_strains2
        (LHS=
                (Yes    (maxtype_resolved))
                (>      (<solidsb>.epsmaxeins*0.9-Versolid.epssmall)          (0.0))
                (IsNot  (<solidsb>.theory_strain)        ("large_sum"))
        )
        (HYPO= need_large_strains)
)

(RULE= need_large_strains1
        (LHS=
                (Yes    (maxtype_resolved))
                (>      (<solidsb>.epsmaxeins-Versolid.epssmall)               (0.0))
                (IsNot  (<solidsb>.theory_strain)        ("large_sum"))
                (>      (<solidsb>.ar_eps90-Versolid.areasmall)                (0.0))
        )
        (HYPO= need_large_strains)
)

(RULE= poss_no_supportd
        (LHS=
                (>      (<solidsb>.romin-Versolid.rotsmall)          (0.0))
                (>      (<solidsb>.romin-0.25*<solidsb>.romax)                 (0.0))
                (Yes    (excessive_displ))
        )
        (HYPO= poss_no_support)
        (RHS=
                (Show   ("kbases/no_supprd.txt")
        )
)

(RULE= poss_no_supportr
        (LHS=
                (>      (<solidsb>.romin-Versolid.rotsmall)          (0.0))
                (>      (<solidsb>.romin-0.25*<solidsb>.romax)                 (0.0))
                (No     (excessive_displ))
        )
        (HYPO= poss_no_support)
        (RHS=
                (Show   ("kbases/no_suppr.txt")
        )
)
```

```
(RULE= poss_no_support
        (LHS=
                (<      ([solidsl].minmin-Vessolid.rosmall)          (0.0))
                (Yes    (excessive_disp))
        )
        (HYPO= poss_no_support)
        (RHS=
                (Show   ("kbases/no_suppd.txt")
        )
)

(RULE= show_may_lstrains
        (LHS=
                (Yes    (may_large_strains))
                (No     (poss_no_support))
        )
        (HYPO= show_may_lstrains)
        (RHS=
                (Show   ("kbases/may_lstm.txt")
        )
)

(RULE= show_need_large_disp
        (LHS=
                (Yes    (need_large_disp))
                (No     (poss_no_support))
        )
        (HYPO= show_need_large_disp)
        (RHS=
                (Show   ("kbases/need_lrgdis.txt")
        )
)

(RULE= show_need_lstm
        (LHS=
                (Yes    (need_large_strains))
                (No     (poss_no_support))
        )
        (HYPO= show_need_lstm)
        (RHS=
                (Show   ("kbases/need_lstm.txt")
        )
)

(RULE= new_solid
        (LHS=
                (Yes    (solid_generator.new_solid))
        )
        (HYPO= solid_generator.n+1)
        (RHS=
                (Let    (solid_generator.new_solid)          (FALSE))
                (CreateObject   ('solid'\solid_generator.n\)          (lsolidsl))
                (Do     (solid_generator.n+1)        (solid_generator.n))
        )
)
```

```
(RULE= sus_mayiarge
        (LHS=
                (Yes    (matype_resolved))
                (is     (<solids> theory_mat)       ("elastic"))
                (>      (<solids> sumaxsis-<solids> sufimei)        (0.0))
                (<=     (<solids> ar_sus90-Vesolid areasmall)       (0.0))
        )
        (HYPO= sus_mayiarge)
        (RHS=
                (Show   ("kbases/sus_mayiarge.txt")
        )
)

(RULE= sus_toolarge
        (LHS=
                (Yes    (matype_resolved))
                (is     (<solids> theory_mat)       ("elastic"))
                (>      (<solids> sumaxsis-<solids> sufimei)        (0.0))
                (>      (<solids> ar_sus90-Vesolid areasmall)       (0.0))
                (No     (poss_no_support))
        )
        (HYPO= sus_toolarge)
        (RHS=
                (Show   ("kbases/sus_toolarge.txt")
        )
)
```